VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Bakalářská práce

2009 **Jan Štambachr**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra měřicí a řídicí techniky

Bezdrátové komunikační rozhraní mezi vestavěnými systémy
Wireless communication interface of embedded systems

2009 **Jan Štambachr**

Prohlášení

„Prohlašuji, že

- *jsem tuto bakalářskou/diplomovou práci vypracoval samostatně.*
- *Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.*
- *jsem byl seznámen s tím, že na moji bakalářskou/diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména §35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a §60 – školní dílo.*
- *beru na vědomí, že Vysoká škola báňská – technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně ke své vnitřní potřebě bakalářskou/diplomovou práci užít (§35 ods. 3).*
- *souhlasím s tím, že jeden výtisk bakalářské/diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a údaje o bakalářské /diplomové práci budou zveřejněny v informačním systému VŠB-TUO.*
- *beru na vědomí, že odevzdáním své práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby.“*

V Ostravě 7. 5. 2009 *Podpis studenta*

Poděkování

Chtěl bych touto cestou velmi poděkovat vedoucímu mé diplomové práce panu Ing. Zdeňkovi Macháčkovi, Ph.D. za cenné rady, konzultace a připomínky spojené s vypracováním mého úkolu.

Abstrakt

Bakalářská práce řeší problematiku bezdrátové komunikace. Jedná se o přenos informací mezi dvěma nebo více zařízeními, které mohou mít jak vysílací, tak přijímací funkci. Nosným médiem není kabel nebo jiné pevné spojení, ale elektromagnetické záření specifikované vlnové délky. S nástupem moderní elektroniky se ve světě čím dál tím víc využívá nesporných výhod tohoto typu komunikace. Bezdrátový přenos dat je v dnešní době hojně využíván jak v průmyslu, tak v komerčním sektoru. Pod pojmem bezdrátová komunikace si také lze představit širokou škálu nejrůznějších standardů rozdílné složitosti, nabízejících rozdílný dosah, bezpečnost, zabezpečení, spolehlivost, rychlost přenosu, schopnost komunikovat v sítích... Práce obsahuje teoretický rozbor výše zmíněné problematiky a návrh a realizaci původních úloh bezdrátové komunikace.

Dissertation solves questions of wireless communication. It is considered to be a transmission of information between two or more devices, which can have transmitting or receiving function. Data carrier is not a cable or any other solid connection, but electromagnetic radiation of specified wave length. Indisputable advantages of wireless communication are being put to use worldwide much more often thanks to upcoming, modern electronics. Nowadays, wireless data transmission is being used frequently both on the field of industry and commercial sector. Conception of wireless communication also evokes a wide spectrum of assorted standards, which varies in complexity, offers different range, safety, security, reliability, transmission speed, ability of net-communication… This work contains analysis on theoretical basis of mentioned points and design and realization of original wireless communication tasks.

Klíčová slova

Bezdrátová komunikace, rádiový modul nRF24LU1, rádiový přenos, Nordic Semiconductors, Keil µVision, Enhanced ShockBurst, cyklická redundantní kontrola (CRC), kryptování, pokročilý kryptovací standard (AES), potvrzovací paket, flash paměť, míra chybovosti paketů, míra chybovosti bitů,, zásobník „první dovnitř, první ven“ (FIFO), hlavní řídící jednotka (MCU).

Wireless communication, RF transceiver nRF24LU1, RF transmission, Nordic Semiconductors, Keil µVision, Enhanced ShockBurst™, cyclic redundancy check (CRC), encryption, advanced encryption standard (AES), acknowledgement packet, flash memory, packet error rate (PER), bit error rate (BER), container “first in, first out” (FIFO), Main control unit (MCU).

Seznam použitých symbolů a zkratek

- ACK - (Acknowledgement) Označení pro potvrzovací paket.
- AES - (Advanced Encryption Standard) Pokročilý kryptovací standard.
- BER - (Bit Error Rate) Míra chybovosti bitů.
- CRC - (Cyclic Redundancy Check) Cyklická redundantní kontrola.
- DES - (Data Encryption Standard) Standard kryptování dat.
- FIFO - (First In, First Out) První dovnitř, první ven. Způsob manipulace s daty. Data jsou obsluhována podle pořadí v jakém byly přijaty.
- GFSK - (Gaussian Frequency-Shift Keying) Gausova frekvenční modulace.
- HID - (Human Interface Device) Zařízení rozhraní člověk – PC. Přijíma povely (vstupy) člověka a může vracet informace (výstupy).
- ISM - (Industrial, Scientific and Medical) Rádiové frekvence používané v oblastech průmyslu, vědy a zdravotnictví.
- ISP - (In-System Programing) Programování vestavěných systému, označuje rozhraní.
- MCU - (Main Control Unit) Hlavní řídící jednotka.
- MDS - (Maximum Distance Separable) Matice reprezentující funkci s jistými rozptylovými vlastnostmi.
- PC - (Personal Computer) Osobní počítač.
- PCB - (Printed Circuit Board) Tištěná deska spojů.
- PER - (Packet Error Rate) Míra chybovosti paketů.
- PID - (Packet Identify) Identifikace paketu, označení pro 2 bity v kontrolním poli paketu.
- PLL - (Phase-Locked Loop) Smyčka fázového závěsu.
- RAM - (Random Access Memory) Paměť s libovolným přístupem.
- RF - (Radio Frequency) Rádiová frekvence.
- SMA - (SubMiniature version A) Miniaturní konektor pro koaxiální kabel.
- UHF - (Ultra High Frequencies) Velmi vysoké frekvence, pásmo 0,3 – 3 GHz.
- USB - (Universal Serial Bus) Universální sériová linka.

Obsah

1 Úvod

Tato bakalářská práce se zabývá bezdrátovou komunikací mezi vestavěnými systémy. Pod tímto si lze představit přenos dat na určitou vzdálenost za pomocí vhodných hardwarových prostředků. Cílem této práce je především tvorba vlastní aplikace pro bezdrátovou komunikaci, ale také teoretické objasnění problémů, vlastností a způsobů řešení a provedení bezdrátového přenosu dat. Aplikace je vyvíjena na vývojovém kitu nRF24LU1 firmy NORDIC Semiconductors. V tomto případě se jedná o přenos binárních dat rádiovými vlnami na malou vzdálenost (do 3 m). Samotná komunikace rádiových modulů firmy NORDIC se nachází v pásmu rádiových vln UHF, 300 – 3000MHz. mezi jednotlivými rádiovými moduly, které vývojová stavebnice obsahuje. Tyto moduly je možné připojit k PC USB rozhraním a toto spojení lze využít jako napájení, nebo také pro komunikaci s PC a vizualizaci přenášených dat. Součástí práce je také vysvětlení problémů spojených s přenosem dat (zejména spolehlivost přenosu) ve vytvořených úlohách.

Práce je rozdělena na dvě hlavní části, teoretickou a praktickou. Teoretická část nahlíží na práci z obecně teoretického hlediska, přináší přiměřený popis a vysvětlení dané problematiky. Teoretická část se skládá z těchto kapitol:

- Bezdrátová komunikace – objasňuje podstatu bezdrátové komunikace a obsahuje stručné vysvětlení nejdůležitějších pojmů týkajících se dané problematiky, jejichž uvedení je pro pochopení práce důležité.
- Vývojový kit nRF24LU1 – přináší bližší pohled na použité hardwarové vybavení použité k vývoji aplikace a jeho části a funkce, které jsou v aplikaci přímo využity.
- CRC – detailně popisuje metodu cyklické redundantní kontroly.
- Kryptování – detailně popisuje problematiku šifrování a využití v bezdrátovém přenosu.

Praktická část práce je zaměřena na samotnou aplikaci bezdrátové komunikace. Kapitola „Praktická realizace základních aplikací bezdrátové komunikace je pojata jako programová dokumentace k původním aplikacím a obsahuje tyto kapitoly:

- Prostředky návrhu – uvádí hardwarové a softwarové prostředky využité v aplikaci a jejich konkrétní využití.
- Komunikace mezi dvěma body – dokumentuje návrh a funkci původní aplikace komunikace dvou bodů.
- Komunikace mezi třemi body – dokumentuje návrh a funkci původní aplikace komunikace tří bodů v sérii a uvádí způsob vizualizace doručených dat prostřednictvím propojení s PC přes USB rozhraní.
- Vyhodnocení míry chybovosti bezdrátového přenosu – v této kapitole jsou původní úlohy v praxi otestovány. Je zjištěna míra chybovosti v závislosti na rostoucí vzdálenosti přenosu a zhodnocena přenosová rychlost v závislosti na různých parametrech.

2 Teoretický úvod do problematiky bezdrátové komunikace

2.1 Bezdrátová komunikace

2.1.1 Základní popis:

Pod pojmem bezdrátová komunikace se rozumí přenos informací na určitou vzdálenost bez použití elektrických vodičů. Vzdálenost může být různá, od několika málo metrů (např. ovládač televize) až po milióny kilometrů (rádiová komunikace s družicemi). Bezdrátová komunikace se obecně považuje za odvětví telekomunikací.

Bezdrátové komunikace se užívá všude tam, kde by bylo nemožné, nebo nepraktické zavádět kabelové vedení. Jako přenosové médium se používá elektromagnetické záření zvolené frekvence (vlnové délky) např. rádiové vlny, infračervené záření, mikrovlny, laser, viditelné světlo atd., nebo jiné formy energie volně přenosné prostorem (např. zvuk).

2.1.2 Historie:

Historie bezdrátové komunikace je neodmyslitelně spojena s vynálezem rádia, jehož vývoj započal jako rádiová telegrafie na konci 19. století. O rozvoj bezdrátové komunikace se na teoretické úrovni zasloužili převážně teoretičtí fyzikové: např. Michael Faraday objevem elektromagnetické indukce, James Clerk Maxwell vytvořením teorie elektromagnetického pole, která předpověděla existenci rádiových vln a Heinrich Rudolf Hertz, který teorii elektromagnetických vln demonstroval v praxi. Mnoho vědců se následně snažilo o praktickou realizaci komunikace na dálku bez použití drátů. Prvním komu se povedlo spolehlivě vytvářet a přijímat rádiové vlny byl Nikola Tesla, který také obdržel patent na vynález rádia.

Následný vývoj směřoval k vynálezům rozhlasového, později televizního vysílání. Rozvoj elektroniky a výpočetní techniky ve 20. století umožnil rozšíření bezdrátových technologií jak do jiných oblastí elektromagnetického spektra, tak do jiných oblastí praktického využití. Z příkladů implementace bezdrátové technologie v běžné praxi, lze mimo již zmíněnou rádiovou komunikaci uvést např. televizní ovládač, bezpečnostní systémy budov, mobilní telefon, WiFi, nebo bezdrátový přenos energie (transformátor). [3] [4]

2.2 Vývojový kit nRF24LU1

2.2.1 Základní popis:

Vývojový kit nRf24LU1, obsahující mimo jiné jednočipový rádiový vysílač/přijímač, byl vyvinut za účelem umožnit uživatelům získat zkušenosti se zařízením umožňujícím rádiofrekvenční bezdrátovou komunikaci a uplatnit jej ve svých budoucích aplikacích. Vývojový kit je vhodný pro použití v programovací části vývoje, pro testování nebo debugging PC softwaru nebo kódu pro MCU.

Vývojový kit nRF24LU1 obsahuje:

Hardware:

- 2 základní desky
- 2 nRF24LU1 moduly s PCB anténami
- 1 nRF24LU1 rédiový modul s SMA konektorem
- 1 USB Dongle vybavený rádiem nRF24LU provedený v referenčním designu
- 1 USB Dongle programovací adaptér
- 1 ISP Dongle
- 1 PS/2 adaptér
- 1 USB Dongle programovací kabel adaptéru

Software:

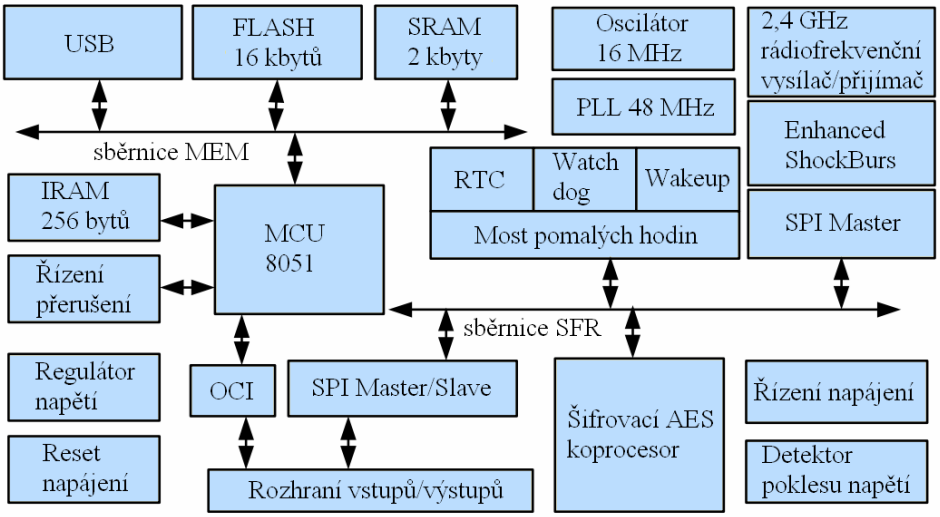
- nRF24LU1 Configuration and Evaluation Suite (CES), v1.0, build 1926.
- nRF24LU1 Flash Programmer application, v1.2, build 1891.
- nRF24LU1 Software Development Kit, v1.2.
- nRF2601 Wireless Desktop Protocol, v.1.1.
- Evaluation version of Keil c51 compiler and µvision IDE

Software může být používán pouze s operačním systémem Windows XP.

Konkrétní využití jednotlivých hardwarových a softwarových komponent v aplikacích bezdrátové komunikace lze nalézt v kapitole č. 3.1 – Prostředky návrhu.

nRF24LU1 obsahuje:

- Rychlý 8 bitový MCU
- 16 kb flash paměť, 2 kb RAM paměť
- 6 programovatelných digitálních I/O
- Výkonný 2.4 GHz RF vysílač/přijímač
- AES kryptovácí hardwarový blok
- USB 2.0
- Funkce ovládání napájení
- Oscilátor a PLL k zajištění plné rychlosti USB operací a redukci potřeby externích komponent
- Generátor resetu napájení a detektor poklesu napájecího napětí
- Podpora hardwarového debuggingu
- Kopletní firmwarová platforma

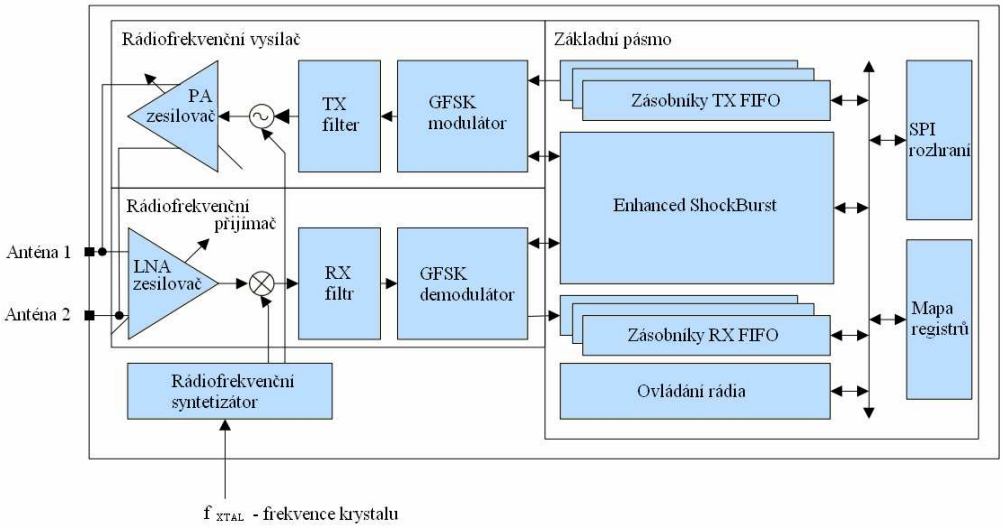


Obrázek č. 1 - Blokové schéma rádiového modulu nRF24LU1.

Následující text pojednává pouze o vlastnostech a funkcích esenciálních pro vývoj aplikace bezdrátového přenosu.

2.2.2 nRF24LU1 RF Transceiver:

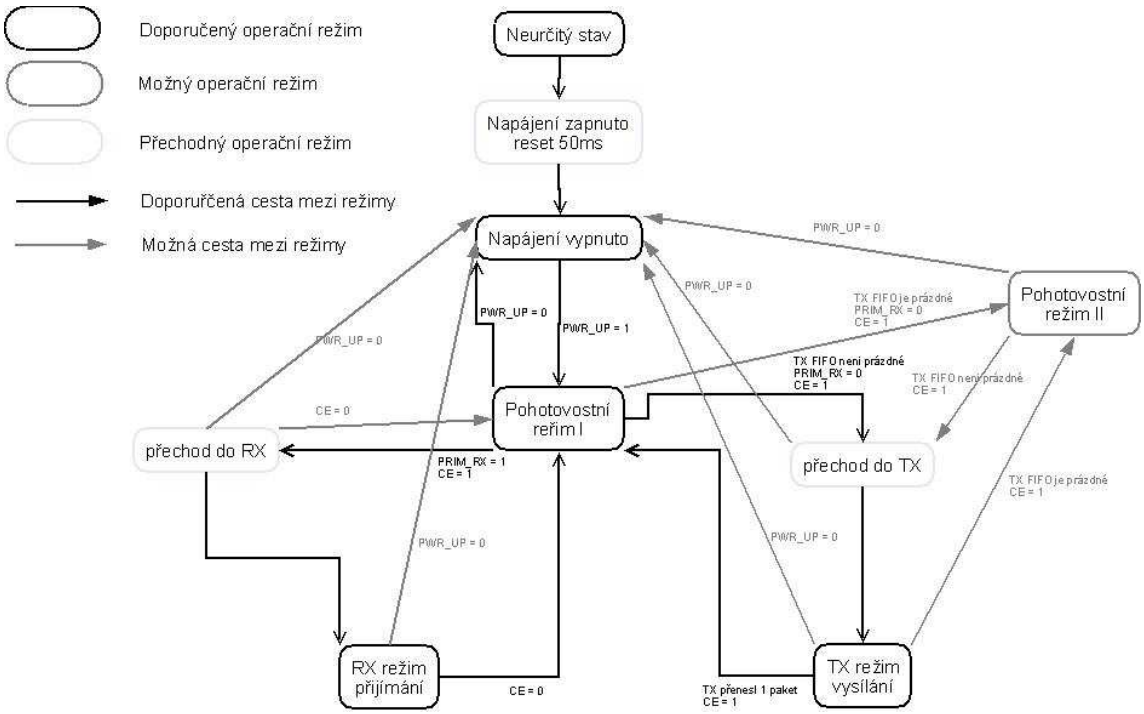
Pro úspěšné zvládnutí tvorby aplikace bezdrátového přenosu je bezpodmínečně nutné seznámit se s principem a funkcemi rádiového vysílače/přijímače. Rádio pracuje v rádiovém pásmu ISM na frekvenci 2,4 – 2.4835 GHz, a využívá vestavěnou linkovou vrstvu Enhanced ShockBurst™, designovanou pro bezdrátové aplikace s nízkými požadavky na energii. Enhanced ShockBurst™ je založen na paketové komunikaci a podporuje nejrůznější módy od manuálního provozu, až po pokročilý autonomní provoz. Vnitřní FIFO organizace zajišťuje hladký tok dat mezi rádiem a systémovým MCU. Rádio využíva GFSK modulaci, která má uživatelsky nastavitelné parametry jako je frekvence kanálu, výstupní energie a hustota vysílaných dat.



Obrázek č. 2 - Blokové schéma rádiového vysílače.

2.2.3 Operační módy:

RF vysílač/přijímač má zabudovaný stavový mechanizmus, který kontroluje přechody mezi jednotlivými operačními módy rádia. Mechanizmus si bere stavy z uživatelem definovaných hodnot registrů a vnitřních signálů. Rádio může být nakonfigurováno do 4 hlavních operačních módů: power down, Standy, TX (vysílací) a RX (přijímací). Stavový diagram ukazuje módy ve kterých může rádio pracovat stejně tak jak je k těmto módů přistupováno, obrázek č. 3.



Obrázek č. 3 - Stavový diagram přechodu mezi operačními režimy.

Power down mód:

V tomto módu je rádio vypnuto s minimální spotřebou proudu, ale všechny hodnoty registrů jsou udržovány a přístupné přes SPI rozhraní. Rádio vstoupí do módu vypnuto nastavením bitu PWR_UP v CONFIG registru na 0.

Standby mód:

Nastavením bit PWR_UP na 1 vstoupí rádio do módu standby-I, který se používá k minimalizaci spotřeby proudu při zachování krátkých časů spouštění. V tomto módu je aktivní pouze část krystalového oscilátoru. Na rozdíl od módu standby-II , při kterém jsou aktivní navíc hodinové buffery a spotřebovává se daleko víc proudu. Pokud je rádiový modul v režimu TR nebo RX a nultý bit v RFCON registru s označením RFCE nastaven na 0, rádio se vrací do režimu Standy-I, pokud je ale tento bit udržován na 1 a TX FIFO zásobník je prázdný, vrací se rádio do módu standby-II. Pokud je nahrán nový paket do zásobníku TX FIFO, spustí se PLL a paket je přenesen.

RX mód:

V tomto režimu má rádio přijímací funkci. Aby rádio vstoupilo do RX módu, musí byt bity PWR_UP, PRIM_RX, RFCE nastaveny na 1. Rádio pak demoduluje signál z RF kanálu a neustále předává demodulovaná data vestavěné hardwarové vrstvě pro správu paketů Enhanced ShockBurst™, která v demodulovaném signálu hledá platný paket. Rádio setrvává v režimu RX dokud jej MCU nepřepne do módu standby-I nebo power down. Pokud je zapnuta automatická správa paketu (Enhanced ShockBurst™) může rádio z RX režimu přejit přímo i do TX módu aby bylo odesláno potvrzení ve formě ACK paketu směrem k primárnímu přijímači.

TX mód:

TX mód je aktivní režim, ve kterém nRF24LU1 vysílá paket. Aby do něj rádio mohlo vstoupit, musí být bit PWR_UP na 1, PRIM_RX na 0, v zásobníku TX FIFO musí být paket pro odeslání a na RFCE bitu se musí objevit kladný puls o délce minimálně 10 μs. Pokud je RFCE bit zakázán, rádio se vrátí do režimu stanbby-I, pokud je umožněn, závisí další situace na stavu zásobníku TX FIFO. Pokud není prázdný, rádio zůstane v režimu TX a odešle další paket, pokud je prázdný, rádio přejde do módu standby-II. Vysílač PLL v TX režimu operuje v otevřené smyčce. Je důležité, aby nebylo rádio v TX módu déle než 4ms. Pokud je umožněn retransmit (znovuodesílání), rádio se nikdy nenachází v TX režimu tak dlouho, aby bylo toto pravidlo porušeno.

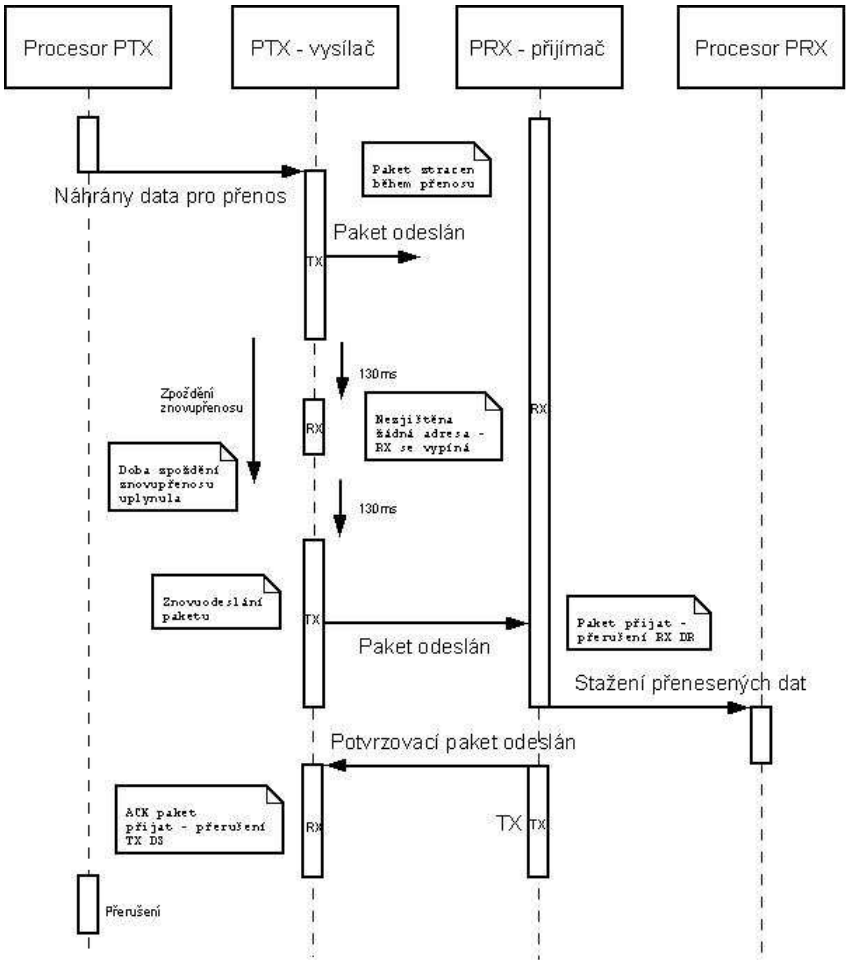
2.2.4 Enhanced ShockBurstTM:

Jedná se o hardwarovou datovou spojovací vrstvu, která má na starosti automatické sestavování, časování, automatické potvrzování a znovuodesílání paketů. Enhanced ShockBurst™ umožňuje výrazné snížení energie potřebné pro komunikaci.

Enhanced ShockBurst™ používá ShockBurst™ pro automatickou manipulaci a časování paketů. Během přenosu ShockBurst™ sestavuje paket a časuje bity v data paketu pro přenos. Během přijímaní ShockBurst™ neustále prohledává demodulovaný signál. Když najde platnou adresu, zpracuje zbytek paketu a ověří jej pomocí CRC algoritmu. Pokud je paket platný, je jeho datová část přesunuta do zásobníku RX FIFO.

Enhanced ShockBurst™ přidává k této automatické správě paketů implementaci spolehlivého obousměrného datového spojení. Jedná se o výměnu paketu mezi dvěma rádiovými moduly kde jeden je primární vysílač (PTX) a druhý primární přijímač (PRX). Výměna paketů je vždy zahájena primárním vysílačem. Přenos je ukončen ve chvíli, kdy PTX obdrží od PRX potvrzovací paket (ACK paket). Tato výměna paketů probíhá takto:

1. Uživatel zahájí transakci přenesením datového paketu z PTX do PRX. Enhanced ShockBurst™ automaticky nastaví PTX do přijímacího režimu a čeká na ACK paket.
2. Pokud PRX přijme datový paket, Enhanced ShockBurst™ nastaví PRX do TX režimu, sestaví a odešle ACK paket do PTX, pak se PRX vrátí do RX režimu.
3. V případě že PTX neobdrží ACK paket do uplynutí přednastaveného časového intervalu, Enhanced ShockBurst™ automaticky znovuodešle původní datový paket a celá procedura se opakuje.



Obrázek č. 5 - Diagram výměny paketů s jedním ztraceným datovým paketem.

Primární přijímač může připojit uživatelem vybraná data k potvrzovacímu paketu a tím umožnit obousměrnou datovou komunikaci. Enhanced ShockBurst™ je široce nastavitelný, lze nakonfigurovat parametry, jako je maximální počet znovuodeslání původní zprávy, prodlevu od odeslání ke znovu odeslání. Toto vše se děje bez zásahu MCU. Funkci Enhanced ShockBurst™ je také možné vypnout, čehož bude využito v aplikaci k vypnutí automatického potvrzování paketů a vyřazení CRC kontroly.

2.2.5 Přerušení:

Jak již bylo naznačeno na diagramu transakce paketů, RF rádio může odesílat požadavky na přerušení pro MCU. Tyto přerušení jsou aktivována pokud jsou nastaveny příslušné bity ve STATUS registru na hodnotu 1. Existují 3 přerušení které může RF rádio odeslat. A to v případě, že:

- Paket je přijat, byl naplněn zásobník RX FIFO. (bit RX_DS)
- Paket je odeslán, v případě že je umožněna funkce Enhanced ShockBurst™ je požadavek na přerušení odeslán až po přijmutí ACK paketu. (bit TX_DS)
- Je dosaženo maximálního počtu znovuodeslání paketu. (bit MAX_RT)

Tyto přerušení jsou přednostně obsloužena hlavní řídící jednotkou. MCU je vektorem přerušení nasměrováno na obslužnou rutinu a tu vykoná. MCU se poté vrátí k vykonávání programu přesně v místě ze kterého bylo nasměrováno obslužné části programu daného přerušení.

V aplikacích hrají přerušení roli synchronizačních prostředků. Přerušení přináší možnost přesně sledovat chvíli, kdy došlo k důležité události (paket doručen, paket ztracen, paket odeslán) a to napomáhá k smysluplnému řízení chodu aplikace. [9] [10] [11]

2.3 CRC

2.3.1 Obecný popis

Cyklický redundantní součet je algoritmus, který z libovolně dlouhého vstupního toku dat vytvoří výsledek o určité délce. Vlastnosti těchto algoritmů může být využito a také velmi často je, při realizaci kontrolního součtu, který je odesílán nebo ukládán spolu se samotnou zprávou nebo daty, v případě že při manipulaci s nimi hrozí jejich poškození nebo ztráta. CRC je populární, protože je velice jednoduché jej implementovat do hardwaru. Na druhou stranu se ale nejedná o absolutní ochranu například před záměrným pozměněním dat. Díky vhodným matematickým vlastnostem není obtížné vypočítat pro pozměněnou zprávu i nový, odpovídající kontrolní součet. Což samo o sobě dělá CRC algoritmy nepoužitelné pro šifrování zpráv. V tom případě je třeba použít složitější hashovací funkce, kterými se bude zabývat kapitola krytování. [5]

2.3.2 Princip CRC

Základní idea v níž CRC spočívá je chovaní se ke zprávě jako k jedinému binárnímu slovu M. Toto binární slovo je vyděleno speciálním klíčovým slovem K, které je známé jak odesilateli tak příjemci. Zbytek po provedení M děleno K je Z, je to v podstatě kontrolní součet zprávy M. Kontrolní součet Z je odeslán spolu se zprávou M příjemci, který opakuje výpočet: provede M děleno K a zkontroluje zda se vypočtený zbytek po dělení rovná příchozímu kontrolnímu součtu. Pokud tomu tak není, Při přenosu zprávy došlo k chybě.

Tato metoda hledání chyb samozřejmě není neomylná existuje mnoho různých zpráv M které při vydělení klíčovým slovem K dají stejný zbytek Z. V tomto případě nebude chyba odhalena. Pravděpodobnost že budou 2 různé zprávy mít stejný kontrolní součet lze minimalizovat zvětšením klíčového slova, to však vede k vysoké redundaci dat. [5]

2.3.3 Výpočet CRC:

Výpočet CRC založen na dělení v konečném tělese GF(2n), zjednodušeně, je to množina polynomů, jejichž koeficienty mohou nabývat pouze hodnot 0 a 1. Každá posloupnost bitů může být přepsána jako polynom, např. posloupnost 111 by dala polynom $x^2 + x + 1$. Tyto polynomy můžeme sčítat, odčítat, dělit nebo násobit stejnějako obyčejné polynomy, např:

$$(x^2 + x + 1) \cdot (x^3 + x + 1) = x^5 + x^4 + 2 \cdot x^3 + 2 \cdot x^2 + 2 \cdot x + 1$$

ale nad výslednými koeficienty se provede operace modulo 2 (zbytek po dělení dvěma). To znamená, že pokud je koeficient sudý přepíšeme jej na 0 a pokud je lichý, tak na 1.

$x^5 + x^4 + 2 \cdot x^3 + 2 \cdot x^2 + 2 \cdot x + 1$ bude $x^5 + x^4 + 1$

Pro ověření, se vydělí $x^5 + x^4 + 1$ původním činitelem $x^2 + x + 1$. To lze provést jednoduše postupnou aplikací bitové operace XOR na dělené číslo podle dělitele, obě reprezentovány bitovým zápisem. Provádí se z leva, vždy na tolik bitů dělence, kolik má bitů dělitel. Po každé operaci XOR se provede rotace dělence doleva o jednu pozici, pokud je na vedoucí bit 1, zapíše se do výsledku 1 a znovu se provede operace XOR podle dělitele, pokud je na vedoucím bitovém místě po rotaci 0, zapíše se 0 a provede XOR podle ekvivalentního počtu nul. Toto se provádí tak dlouho dokud na všech místech dělence nejsou nuly. Výpočet vypadá takto:

```
110001 = 1011
111
001001
 000
001001
  111
000111
   111
000000
```

V tomto jednoduchém výpočtu dělenec reprezentuje bitové slovo, dělitel 111 je bitová reprezentace CRC polynomu a výsledek je kýžený kontrolní součet. [6]

2.3.4 Implementace v nRF24LU1

CRC detekce chyb je součástí Enhanced ShockBurst™. To známenená, že probíhá spolu s automatickou obsluhou paketů, kterou má právě Enhanced ShockBurst™ na starosti. Poté co RF rádio v RX režimu přijme paket, je provedena CRC kontrola adresy paketu, kontrolní oblasti paketu a datového obsahu. Pokud kontrola selže, nelze přijatý paket použít, RX modul neodešle ACK paket a TX modul musí data znovu odeslat (viz. kapitola 2.1.4). Velikost kontrolního součtu může být pro přenos nastavena na 0 (CRC vypnuto), 1 nebo 2 byty. Polynom pro jedno-bytový CRC je $x^8 + x^2 + x + 1$ (CRC-8 ATM), pro dvou-bytový CRC se používá polynom $x^{16} + x^{12} + x^5 + 1$ (CRC-16 CCITT). Údaj o kontrolním součtu zaujímá v paketu poslední 0 – 2 byty. [9]

2.4 Kryptování

2.4.1 Historie kryptování

V kryptografii je kryptování proces transformování informace (prostého textu) použitím algoritmu nazývaného šifra, tak, aby byla nečitelná pro všechny, kromě těch, kteří vlastní určitou vědomost o šifře, obvykle nazývané klíč. Výsledkem operace kryptování je zašifrovaný text.

Historie kryptografie a kryptování sahá tisíce let zpět. Až do posledních desetiletí se jednalo o takzvanou klasickou kryptografii, která využívala metod krytování založených výhradně na použití tužky a papíru, maximálně jednoduchých mechanických pomůcek. Na počátku 20. století přispěl vynález komplexních mechanických a elektromechanických strojů (jako byl za 2. světové války rotorový stroj Enigma) k rozvoji sofistikovanějších způsobů kryptování. A následné zavedení elektroniky a výpočetní techniky umožnilo vypracovat schémata kryptování stále větší komplexnosti, z nichž se většina pro papír a tužku naprosto nehodí.

Kryptování bylo po dlouhou dobu záležitostí výhradně vládních a vojenských organizací k zajištění bezpečné komunikace. Ale v dnešní době je kryptování používáno k ochraně informací v mnoha civilních systémech, jako jsou počítače, úložné jednotky (USB flash disky), bankomaty, internet, mobilní telefony, bluetooth atd. Obecně se dá říct, že kryptování nachází velmi široké uplatnění v bezdrátovém přenosu dat.

Éra moderní kryptografie začíná v 70. letech 20. století a to publikací návrhu Data Encryption Standard (standard kryptování dat, dále jen DES) ve federálním registru Spojených států amerických. Navrhovaná DES šifra byla předložena výzkumnou skupinou firmy IBM jako snaha o vývoj bezpečných zařízení pro elektronickou komunikaci pro obchodní záležitosti jako jsou banky a další velké finanční organizace. Po lehké modifikaci organizací NSA (National Security Agency) byla DES šifra přijata a publikována jako Federální standart pro zpracování informací v roce 1977. Byla to první veřejně přístupná šifra posvěcená velkou národní agenturou, jakou je NSA. To mělo za následek explozivní růst zájmu o kryptografii a kryptování mezi veřejností i v akademických kruzích.

V roce 2001 byl zastaralý a během let několikrát prolomený DES nahrazen standardem AES. Standard AES je v dnešní době nejpopulárnější algoritmus používaný v kryptografii se symetrickými klíči. [7]

2.4.2 AES

Pokročilý standart kryptování AES přijatý vládou USA v roce 2001 jako první šifra schválená pro „top secret“ komunikaci se skládá ze 3 blokových šifer. AES-128, AES-192 a AES-256 převzatých z šifry Rijndael, původně vyvinuté belgickými kryptografy Joanem Daemenem a Vincentem Rijmenem, která byla vybrána v AES výběrovém procesu jako nejlepší. Každá AES šifra má jednotnou, fixní délku najednou zpracovatelného bloku a to 128 bitů a velikost klíče 128, 192 a 256 podle typu šifry. AES algoritmus je rychlý co se týče softwaru i hardwaru. Je jednoduché ho implementovat a spotřebovává málo místa paměti.

Na rozdíl od svého předchůdce DES, který využíval šifru Feinstel AES šifra je síť substitučně-permutační. Tyto sítě se skládají z S-bloků a P-bloků které transformují bloky vstupních bitů na

výstupní bity. Pro tyto transformace je běžné, že jsou složeny z operací, které se efektivně dají provádět na hardwarové úrovni, např. XOR, nebo rotace bitů.

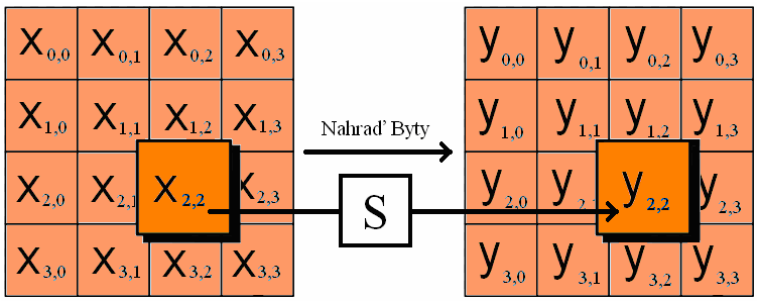
Pakliže má jeden byte 8 bitů, má 128 bitový blok 16 bytů. AES operuje na poli 4 * 4 bytů, nazývaných jako stav. AES šifra je určena jako určité množství opakování transformačních kol, které převádějí vstupní prostý text na finální výstupní šifrovaný text. Každé kolo se skládá z několika kroků, včetně jednoho, který závisí na kryptovacím klíči. Na zpětné převedení šifrovaného textu zpátky na prostý text je použita sada transformačních kol v obráceném pořadí, využívajících stejný klíč.

Popis algoritmu:

- Rozšíření o klíč využívající Rijndael's key schedule – tento algoritmus rozloží krátký klíč na určitý počet kruhových klíčů.
- Úvodní kolo:
  - AddRoundKey – každý byte stavu je zkombinován s kruhovým klíčem.
- Kola:
  - SubBytes – nelineární nahrazování každého bitu za jiný podle vyhledávací tabulky.
  - ShiftRows – transpozice (přemístění) řad stavu o určitý počet bitů.
  - MixColumns – operace, která kombinuje 4 byty v každém sloupci
  - AddRoundKey
- Finální kolo (bez MixColumns):
  - SubBytes
  - ShiftRows
  - AddRoundKey

Krok SubBytes:

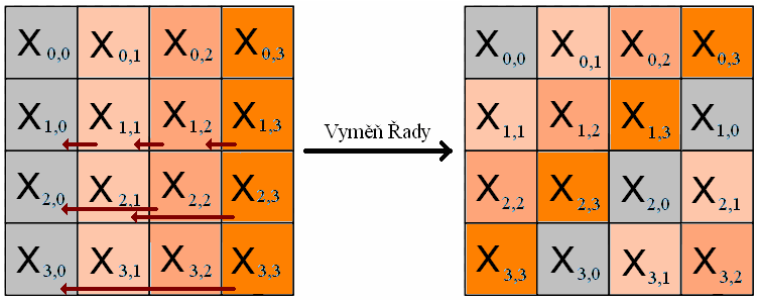
Každý byte v poli je v tomto kroku je nahrazen pomocí 8 bitového substitučního bloku (Rijndael S-blok). Tato operace zajišťuje nelinearitu v šifře. S-blok je odvozen z konečného Galoisova pole GF(2^8), které má dobré nelineární vlastnosti, pomocí operace převrácená hodnota. Aby bylo zabráněno útokům založených na jednoduchých algebraických vlastnostech je S-blok zkonstruován kombinováním inverzní funkce s obrácenou afinní transformací. S-blok je také vybrán, aby se zabránilo vzniku fixních bodů a také jakýchkoli obrácených fixních bodů.



Obrázek č. 6 – Krok SubBytes.

Krok ShiftRow:

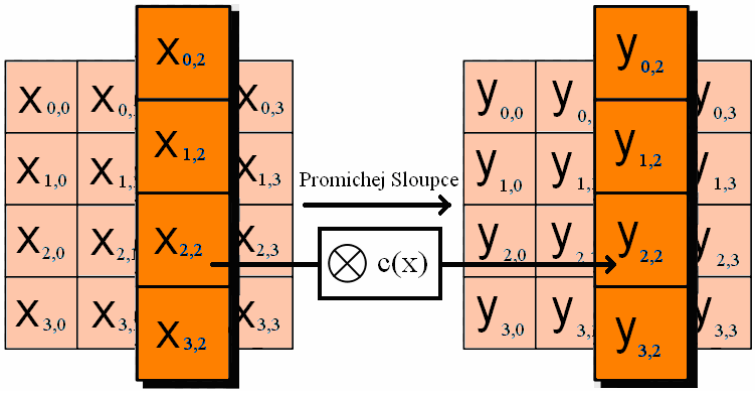
Tento krok cyklicky posouvá byty v každé řadě o určitou hodnotu. U AES algoritmu je první řada nezměněna Každý byte 2. řady je posunut o jednu pozici doleva, analogicky je 3. a 4. řada posunuta o 2 a 3 byty doleva.



Obrázek č. 7 – Krok ShiftRow.

Krok MixColumns:

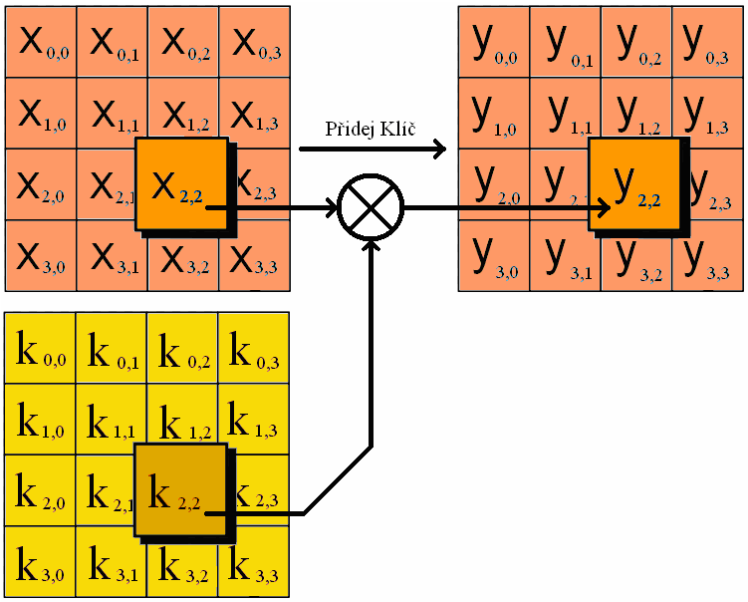
V tomto kroku jsou každé 4 byty každého sloupce zkombinovány použitím převrácené lineární transformace. Každý ze 4 vstupních bytů ovlivňuje všechny 4 výstupní byty. Spolu s funkcí ShiftRows zajišťuje funkce MixColumns v šifře rozptyl. S každým sloupcem je zacházeno jako s polynomem nad $GF(2^8)$ a následně je na něm provedena operace modulo $x^4 + 1$ polynomem $c(x) = 3x^3 + x^2 + x + 2$. N krok MixColumns je také možno nahlížet jako na násobení MDS maticí (matice reprezentující funkci s určitými rozptylovými vlastnostmi) v konečném poli.



Obrázek č. 8 – Krok MixColumns.

Krok AddRound:

V tomto kroku je z hlavního klíče klíč odvozený pomocí metody Rijndael‘s key schedule zkombinován s stavem. Každý odvozený klíč má stejnou velikost jako stav. Odvozený klíč je přidán ke stavu tak, že na každý byte stavu je použita bitová operace XOR podle příslušného bytu klíče.



Obrázek č. 9 – Krok AddRound.

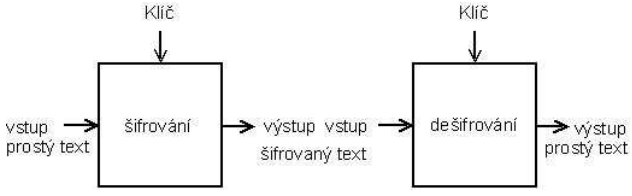
[8]

2.4.3 Implementace AES v nRF24LU1

Mokrokontrolér nRF24LU1 má vlastní, zabudovanou kryptovací jednotku využívající AES algoritmus. AES jednotka podporuje 4 režimy šifrování: ECB, CBC, CFB, OFB a CTR. Všechny využívají 128 bitový klíč a je možnost navolit i 128 bitový inicializační vektor.

Režim ECB – Electronic Code Book (Elektronická kódovací kniha):

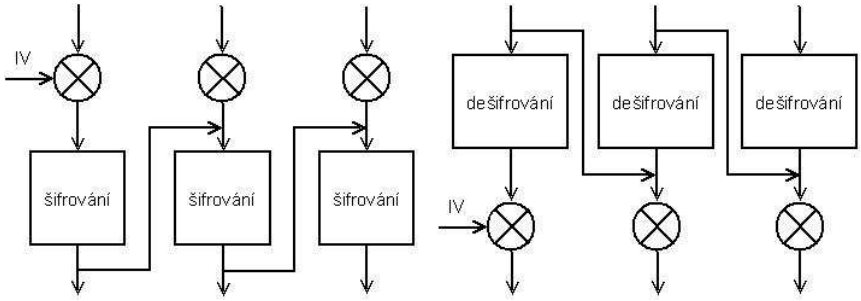
Toto je základní režim šifrování. Prostý text ze vstupu je zašifrován v šifrovacím bloku a poslán na výstup. Při dešifrování je postup opačný, v dešifrovacím bloku je zašifrovaný text pomocí klíče převeden zpátky na text prostý. ECB procedura musí k dešifrování použít poslední rozšířený klíč. Dešifrovací funkce je k funkci šifrování identická, pouze jednotlivé operace probíhají v opačném pořadí (obrázek č. 10).



Obrázek č. 10 – Blokové schéma šifrování ECB.

Režim CBC – Cipher Block Chaining (Řetězení šifrovacích bloků):

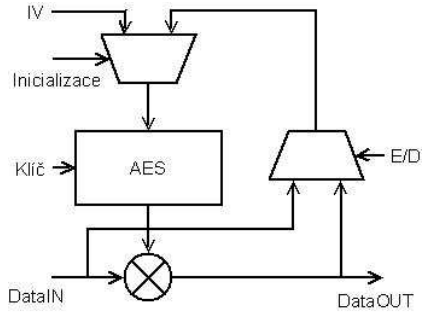
CBC režim přidává mechanizmus zpětné vazby do blokové šifry. Výsledek předchozího šifrování je podroben operaci XOR s příchozími daty. Inicializační vektor IV je použit pro první opakování. Pro dešifrování používá CBC poslední rozšířený klíč. Dešifrovací funkce je s šifrovací identická, pouze převrací pořadí šifrovacích operací (obrázek č. 11).



Obrázek č. 11 – Blokové schéma šifrování CBC.

Režim CFB – Cipher FeedBach (Zpětná vazba šifry):

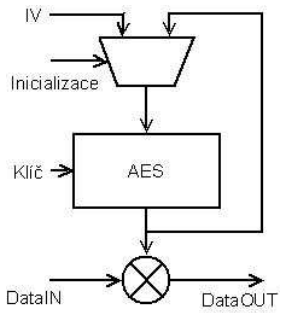
Výsledek šifrování v režimu CFB je odeslán zpět do vstupu AES bloku. Pro první opakování (kolo) je použit iterační vektor IV. Vstupní data jsou šifrována provedením bitové operace XOR s výstupními daty šifrovacího bloku. Dešifrovací funkce je s šifrovací identická, pouze převrací pořadí šifrovacích operací (obrázek č. 12).



Obrázek č. 12 – Blokové schéma šifrování CFB.

Režim OFB - Output FeedBack (Zpětná vazba na výstupu):

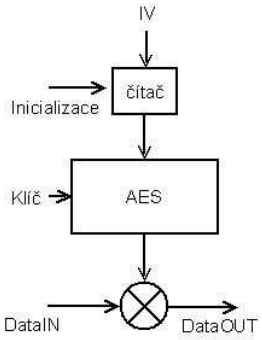
Tento režim je podobný režimu CFB, liší se pouze v logických úpravách výstupních dat, než jsou odeslána zpět na vstup (obr č. 13).



Obrázek č. 13 – Blokové schéma šifrování OFB.

Režim CTR – Counter (Čítač):

V tomto režimu je na vstup AES bloku je připojen čítač. Pro inicializaci čítače je použit inicializační vektor IV. Vstupní data jsou šifrována provedením bitové operace XOR s výstupem šifrovacího modulu. Dešifrovací funkce je s šifrovací identická, pouze převrací pořadí šifrovacích operací (obrázek č. 14).



Obrázek č. 14 – Blokové schéma šifrování CTR.

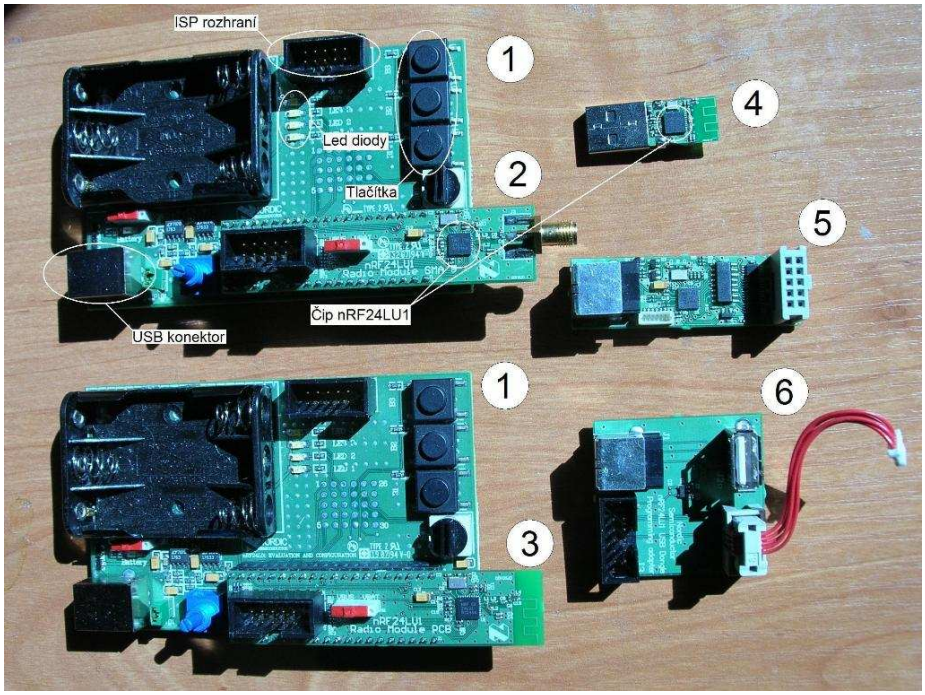
Šifrování je řízeno AESCS registrem, kde kombinace 2 – 4 bitu volí režim šifrování (viz. výše), 1. bit volí mezi enkrypcí a dekrypcí a nultý bit odstartuje danou operaci. Po ukončení operace je vytvořen požadavek na přerušení a MCU přečte AESD registr, ve kterém se nachází výsledek operace šifrování. Pro vložení dat pro en/dekrypci slouží AESKIN registr. Oba tyto registry jsou 128 bitové a data jsou v nich uložena ve formě dvojdimenzionálních polí. Zápis nebo čtení z nich je ale možné provádět jen po 8 bitech. Pro vyčtení nebo zápis každé pozice v poli je třeba provést 16 za sebou jdoucích operací čtení, nebo zápisu. Každá tato operace inkrementuje dva 4 bitové ukazatele na pozici pole. Tyto ukazatele se nachází v AESIA1 registru. [9]

3 Praktická realizace základních aplikací bezdrátové komunikace

3.1 Prostředky návrhu

3.1.1 Hardware

Úlohy jsou tvořeny pro vývojový kit nRF24LU1, viz obrázek č. 15. Základní požadavek na aplikaci je komunikace mezi dvěmi a více komunikačními body. Z vývojového kitu jsou pro základní aplikaci komunikace mezi dvěma body využity nRF24LU1 rádiové moduly SMA a PCB umístěné na základních deskách vybavených čtyřmi led diodami a třemi tlačítky. Digitální vstupy a výstupy (tlačítka, led diody) byly využity k ladění při vývoji aplikace, dále jsou využívány k diagnostice a ovládání. Pro hlavní aplikaci bezdrátové komunikace mezi třemi body funkci prostředního komunikačního bodu plní nRF24LU1 USB dongle. Pro programování všech těchto rádiových modulů je použit ISP programátor a adaptér pro spojení s USB single, rovněž na obrázku č. 15.



Obrázek č. 15 – Vývojový kit nRF24LU1

1. Základní deska – Obsahuje zdroj energie, držák pro umístění baterií, konektory (USB, ISP), led diody, tlačítka. Tyto komponenty umožňují využívat základní vlastnosti vývojového kitu. Tlačítka a led diody jsou v úlohách využity k monitorování a řízení chodu programu.
2. SMA rádiový modul – Je připojen k základní desce pomocí 40 pinového dvouřadého konektoru. Obsahuje rádiový čip nRF24LU1 a má zabudovaný SMA konektor pro

koaxiální kabel, který dovoluje připojit externí anténu. V úlohách je využíván jako koncový přijímací komunikační bod.

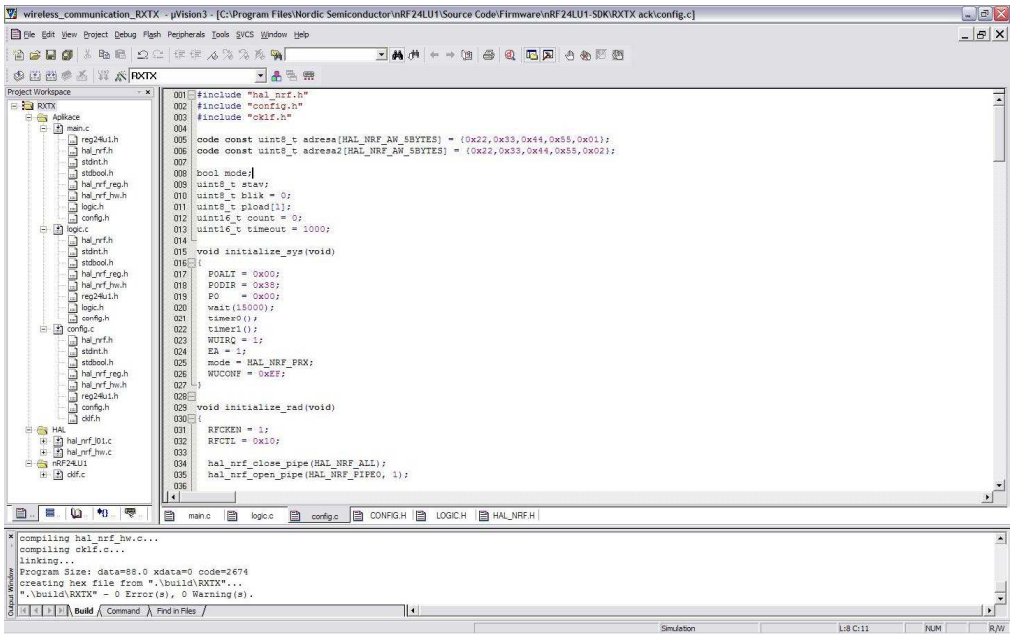
3. PCB rádiový modul – Na rozdíl od SMA modulu, obsahuje PCB anténu. Anténa je zakomponována v integrovaném obvodu. V úlohách je využíván jako vysílací komunikační bod.
4. USB dongle – Miniaturní verze rádiového modulu s PCB anténou. Má stejné vlastnosti a funkce. Pomocí USB konektoru je možné jej připojit k PC. V úloze komunikace tří bodu je využit jako prostřední komunikační článek.
5. ISP programátor – Slouží k nahrávání programu do flash paměti rádiového modulu. Má USB konektor (připojení k PC) a k základní desce vývojového kitu se připojuje přes ISP rozhraní.
6. Adaptér pro USB dongle – Propojuje USB dongle s ISP programátorem. [9] [10] [11]

3.1.2 Vývojové prostředí

Aplikace pro řízení rádiových modulů je vytvořena ve vývojovém prostředí Keil µVision, který umožňuje programovat mikrokontroléry v jazyce C. Prostředí umožňuje po přeložení z kódu vytvořit soubor typu .hex, kterým již lze pomocí nRF24LU1 flash programátoru od firmy Nordic naprogramovat flash paměť rádiových modulů. Pro nahrání programu do flash paměti slouží ze strany PC software firmy NORDIC nRF24LU1 Flash Programmer, obrázek č. 16.

Obrázek č. 16 – nRF24LU1 Flash Programmer.

Zakládání projektu pro vývoj aplikace vyžaduje výběr hardwaru pro který bude generovaný kód sloužit. Vývojové prostředí Keil µVision podporuje tvorbu programů pro mnoho různých typu mikrokontrolérů. Aplikace pro bezdrátovou komunikaci je vytvořena pro mikrokontrolér nRF24LU1 od firmy Nordic Semiconductor. Dále je provedeno nastavení vlastností projektu (Options for targer „jméno projektu“). V záložce Output je zaškrtnuto vytváření HEX souboru. V záložce C51 je třeba do řádku „Include paths“ zadat cestu v systému souborů na disku k souborům potřebným k implementaci do projektu. Potřebné .c soubory a jejich hlavičkové soubory lze implicitně nalézt v podložkách adresáře C:\Program Files\Nordic Semiconductor\nRF24LU1\Source Code\Arch. Toto nastavení je také možné provést pro každý projektový .c soubor zvlášť.



Obrázek č. 17 – Vývojové prostředí Keil µVision.

3.2 Komunikace mezi dvěma body

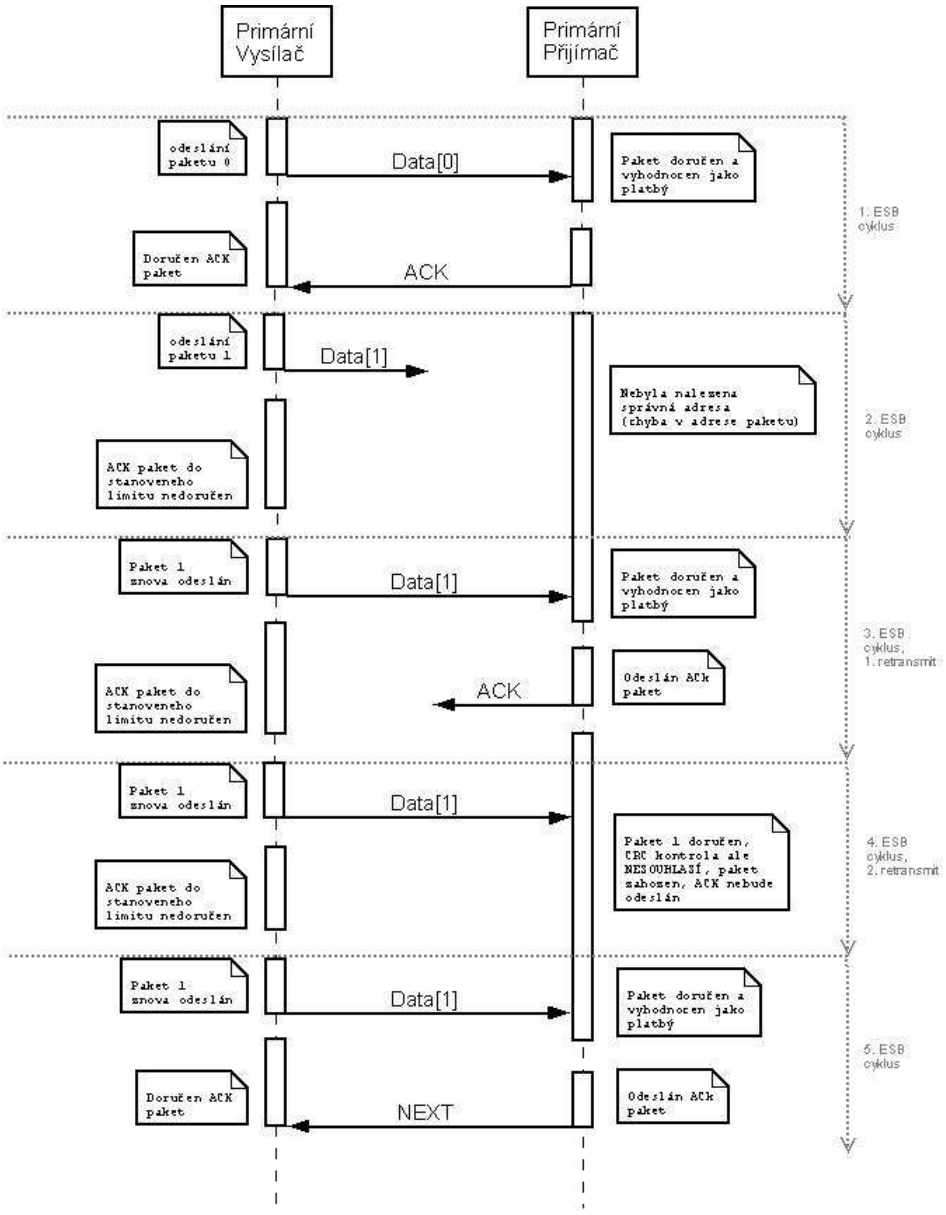
3.2.1 Požadavky

Původní aplikace pro komunikaci mezi dvěma body slouží k seznámení se s základními parametry rádiových modulů, dále slouží k testování a ladění programových struktur, která jsou uplatněny v hlavní úloze. Aplikace využívá digitální vstupy a výstupy, využívá časovače, RF přerušení. Hlavním požadavkem na aplikaci komunikace dvou rádiových modulů je možnost za běhu programu, bez nutnosti kompletního restartu MCU, měnit parametry ovlivňující charakter přenosu. Druhým požadavkem je optimalizace algoritmu který se zabezpečuje odesílání paketů, tak aby nedocházelo ke zbytečným časovým prodlevám. Aplikace je také využita pro testování přenosu s různě nastavenými parametry, získané poznatky jsou uplatněny pro návrh aplikace sériové komunikace tří bodů.

3.2.2 Návrh komunikačního systému

Rádiový modul může pracovat v plně automatickém režimu, řízeným platformou Enhanced ShockBurst^{Tm}. Pokud je Enhanced ShockBurst vypnut, není možné využít funkce automatického potvrzování paketů. V praxi to znamená, že rádio vysílá pakety kontinuálně bez jakékoli zpětné vazby od přijímače. Stejně tak přijímač tuto zpětnou vazbu neposkytuje. Pokud ovšem tato situace není ošetřena samotným programem, což může být pro některé situace výhodnější. Automatické potvrzování je jeden z parametrů které lze za chodu aplikace vypnout nebo zapnou. Další konfigurační parametry s automatickým potvrzováním úzce souvisí, je to například datová šířka přenášených dat, typ požitého CRC, počet znovuodeslání nedoručeného paketu, doba čekání na potvrzovací paket, frekvence přenosu a rychlost datového toku. Jelikož nastavení obou komunikujících modulů musí být

totožné, Je třeba při změně parametrů u jednoho modulu, zajistit jejich změnu i u druhého. V aplikaci je toto řešeno vysláním konfiguračního paketu z vysílacího modulu do modulu přijímacího, který po přijetí provede zadané změny nastavení. Dané parametry zadává uživatel pomocí tlačítek na základní desce vysílače (bližší specifikace v kapitole č. 3.2.6). Na časovém diagramu (obrázek č. 18) je zachycen chod úlohy komunikace 2 bodů. Data proudí z primárního vysílače do primárního přímače, který příjem potvrzuje odesíláním ACK paketů. Konkrétně je na obrázku zobrazeno 5 Enhanced ShockBurst cyklů, postihujících všechny chybové možnosti, které mohou nastat (ztráta datového paketu, ztráta ACK paketu, chybná CRC kontrola).



Obrázek č. 18 – Chod úlohy komunikace 2 bodů v čase.

Problematika optimalizace odesílání paketů z vysílače může být řešena dvěma způsoby, které se liší podle doby udržování bitu RFCE na hodnotě 1. Pokud je tento bit (nultý bit RFCON registru) nastaven na 1 a v zásobník TX FIFO není prázdný, je odeslán paket (viz. kapitola 2.2.3). Oba tyto způsoby nastavování RFCE bitu se hodí pro jinou situaci.

Logika prvního způsobu vysílání nejprve data pro odeslání přesune do zásobníku TX FIFO a poté je na bitu RFCE vytvořen 10 µs impuls. Po odeslání program čeká na RF přerušení (TX_DS -

paket doručen nebo TX_MAX_RT – maximální počet znovupřenosů). Pokud byl paket úspěšně doručen je automaticky TX FIFO vyprázdněn, pokud doručen nebyl při přerušení TX_MAX_RT je programem smazán. Následně se program vrací do hlavní smyčky.

NA rozdíl od druhého způsobu, kde je nejprve nastaven RFCE bit na 1 a poté jsou přesunuty do TX FIFO první data pro odeslání. Program čeká na přerušení, jakmile jedno z možných přerušení nastane, nová data jsou přesunuty do TX FIFO.

Tento druhý způsob, na rozdíl od prvního nevyužívá pulsů na bitu RFCE, ale po dobu přenosu všech paketů je udržován programem na hodnotě 1. Pro velké množství po sobě přenášených paketů v tento způsob výhodnější, jednak šetří počet prováděných operací mezi přenosy, navíc rádio nemusí vstupovat do režimu standby – I, ale po dobu kdy je prázdné TX FIFO přechází do režimu standby – II což sebou nese 130 µs časovou úsporu (viz. obrázek č. 3). Pro komunikaci bez automatického potvrzování je ale tento způsob nevhodný. Rádiový modul se smí nacházet ve vysílacím režimu nejdéle 4 ms, toto pravidlo by muselo být ošetřeno v programu dodatečnými podmínkami. Pokud je ale automatické potvrzování zapnuto Enhanced ShockBurst^{Tm} přepíná rádio pro přijetí potvrzovacího paketu do přijímacího režimu automaticky v dostatečně krátkých časových intervalech. V praxi, kdy jsou funkce platformy Enhanced ShockBurst^{Tm} (automatické potvrzování paketů) využívány, jsou oba způsoby téměř rovnocenné.

3.2.3 Struktura aplikace

Samotný projekt vytvořený v prostředí Keil µVision je složen z několika souborů typu .c, které jsou rozděleny do 3 složek. První složka s názvem Aplikace obsahuje 3 .c soubory:

- main.c – hlavní soubor, využívající funkce z ostatních dvou .c souborů ve funkci main. Funkce main také obsahuje nekonečnou smyčku while, která zajišťuje cyklické opakování vložených funkcí obsluhujících rádiovou komunikaci.
- logic.c – tento soubor obsahuje funkce obsluhující logiku vysílání a přijímání paketů a zabezpečují interakci s uživatelem pomocí digitálních vstupů (tlačítka) a výstupů (led diody).
- config.c – v tomto souboru jsou funkce zaobalující konfiguraci systémových registrů, které je nezbytné na startu aplikace provést, dále obsahuje funkce obsluhující všechna využívaná přerušení a různé další pomocné funkce.

Součástí projektu jsou i hlavičkové soubory logic.h a config.h. Soubory obsahují definice elementárních funkcí a konstant, které se v projektu často objevují. Tím je docíleno částečného zpřehlednění kódu.

Přesná struktura této hlavní části projektu je znázorněna na obrázcích č. 21, 22, 23 v kapitole 3.2.3 – Struktura aplikace.

Další dvě projektové složky s názvem HAL a nRF24LU1 obsahují .c soubory dodané firmou Nordic Semiconductors spolu s vývojovým kitem.

- hal_nrf_l01.c – Tento soubor obsahuje prototypy funkcí zaobalující zápisy a čtení do nejdůležitějších registrů, které slouží pro nastavování parametrů a řízení rádiového modulu. Tyto funkce jsou přístupné skrze hlavičkový soubor hal_nrf.h.

- hal_nrf_hw.c – Obsahuje funkci, která doplňuje řízení SPI rozhraní, tato funkce je hardwarově závislá, její použití je nutné s mikrokontrolérem nRF24LU1.
- cklf.c - Tento soubor obsahuje prototypy funkcí, které umožňují práci s nízkofrekvenčními hodinami reálného času. Tyto funkce jsou přístupné skrze hlavičkový soubor cklf.h.

3.2.4 Konfigurace registrů rádiového modulu

Kontrolní registry se nacházejí na specifikovaných adresách v paměti. K usnadnění práce s nimi je použit firmou Nordic přiložený hlavičkový soubor reglu24.h který registry mapuje a definuje jejich názvy. Některé názvy odkazují i na jednotlivé bity v registrech. Dále budou uváděny pouze tyto názvy.

Konfigurace MCU

V programu jsou zavedeny dvě funkce, které zastřešují konfigurační zápisy do registrů. První funkce se jmenuje „initialize_sys“ je typu void a nemá žádné parametry. Funkce obsahuje tyto důležité příkazy:

```
P0ALT = 0x00;
P0DIR = 0x38;
P0    = 0x00;
```

Tyto příkazy konfigurují nastavení digitálních vstupů a výstupů. MCU nRF24LU1 má 6 digitálních GPIO (General purpose input/outpu) pinů, nichž každý může být nastaven buď jako vstup, nebo jako výstup. Konkrétně nula zapsaná do P0ALT registru dává pinům základní funkci, jednička by znamenala alternativní funkci (například vstup z časovače, viz. Product_specification_nRF24LU1.pdf). Prvních 6 bitů registru P0DIR určují zda budou jednotlivé piny vstupy nebo výstupy, 7. a 8. bit se nepoužívá. 1 znamená vstup, 0 výstup. Hexadecimální zápis 0x38 (v binárním kódu 111000) znamená že piny 0 – 2 budou výstup (led diody) a piny 3 – 5 vstup (tlačítka). Registr P0 uchovává aktuální hodnotu pinů, v tomto místě kódu je u všech hodnota nastavena na 0.

Další příkaz se týká nastavení přerušení:

```
EA = 1;
```

Zápis jedničky do registru EA znamená povolení globálního přerušení. Pokud by bylo třeba v aplikaci využívat funkci „wakeup“ bylo by třeba podle manuálu nastavit registry WUIRQ a WUCONF.

Funkce „initialize_sys“ dále obsahuje inicializaci časovačů.

Kód vypadá takto:

```
TL0 = 0xe9;
TH0 = 0xcb;
TMOD |= 0x01;
TF0 = 0;
```

```
ET0 = 1;
TCON = 0;
TR0 = 0;
```

Šestnácti bitová hodnota časovače je zapsána v dvou bytových registrech TLx a THx (L – least significant byte, H – most significant byte). V tomto případě je registr nastaven na hodnotu 52201, to znamená že do naplnění časovače je třeba provést 13335 inkrementací. Inkrementace proběhne každých 12 hodinových cyklů, což při frekvenci 16 MHz dává interval 10 ms. Registr TFx uchovává informaci zda nastalo přerušení, při inicializaci je vynulován. Registr ETx při zápisu 1 přerušení časovače povoluje. Registr TCON kontroluje časovač. TMOD registr určí operační mód časovače, v tomto případě mód 1 – 16 bitový časovač.

Konfigurace rádia

Při konfiguraci rádia jsou na rozdíl od přímého zápisu do registrů využívány funkce které tyto zápisy umožňují. Tyto funkce jsou definovány ve firmou Nordic přiloženém .c souboru hal_nrf_l01.c a jejich prototypy se nalézají v hlavičkovém souboru hal_nrf.h.

Nejprve je třeba inicializovat SPI rozhraní rádia. SPI (Serial Programable Interface) je programovatelná sériová sběrnice rádiového modulu. Její nastavení se provede příkazy:

```
RFCKEN = 1;
RFCTL = 0x10;
```

RFCKEN je druhý bit registru RFCON, který kontroluje provoz rádia. Zápis jedničky do jeho druhého bitu znamená povolení radiofrekvenčních hodin (16 MHz). Zápis do 0. až 3. bitu RFCTL registru určuje frekvenci SPI rozhraní. V tomto případě je nastavena na jednu polovinu hodinových pulsů. Jednička na 4. bitu SPI rozhraní zapne.

Další nastavení provádí tyto funkce:

```
hal_nrf_close_pipe(HAL_NRF_ALL);
hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

hal_nrf_set_crc_mode(HAL_NRF_CRC_8BIT);
hal_nrf_set_auto_retr(250, 3);
```

Tento sled příkazů nejprve uzavře všechny komunikační potrubí, následně otevře komunikační trubku 0. Jednička v argumentu znamená zapnutí automatického potvrzování paketu – zapnutí Enhanced ShockBurst™. Dále je nastaven režim CRC, na výběr je buď CRC vypnuto – argument 0, 8 bitové CRC, nebo 16 bitové CRC. Poslední příkaz určí dobu prodlevy před zahájením znovuodeslání nedoručeného paketu, druhý argument určí maximální počet těchto znovuodeslání. Pokud se u rádia počítá pouze s přijímací funkcí, je poslední příkaz zbytečný.

Dále je třeba nastavit identifikační adresu, která je odesílána s každým paketem, popřípadě adresu kterou modul při přijmu očekává.

```
hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
hal_nrf_set_address(HAL_NRF_TX, adresa);
hal_nrf_set_address(HAL_NRF_PIPE0, adresa);
```

Adresa je v tomto případě nastavena na pěti bytovou šířku (lze zvolit i 3 byty). Adresa pro vysílání a příjem je nastavena na předem definovanou konstantu s názvem „adresa“, která je definována jako pole o pěti prvcích s datovým typem unsigned char.

Dále se provádí nastavení závislé na tom jaký operační režim má být po startu spuštěn, pro režim primární vysílač je nastavení následující:

```
hal_nrf_set_operation_mode(HAL_NRF_PTX);
```

Pro primární příkaz je třeba ještě specifikaci na kterém komunikačním potrubí má rádio očekávat zprávu a jaké bytové šířky bude její datový obsah (1 – 32 bytu).

```
hal_nrf_set_operation_mode(HAL_NRF_PRX);
hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
```

Na závěr následují příkazy:

```
hal_nrf_set_rf_channel(channel);
hal_nrf_set_power_mode(HAL_NRF_PWR_UP);
RF = 1;
```

První příkaz určí komunikační frekvenci v rozsahu od 2,4 GHZ do 2,525 GHz, parametr „channel“ je frekvence v jednotkách MHz, která se přičte k implicitní hodnotě 2400 MHz a určí výslednou komunikační frekvenci. Následující příkaz zapne napájení rádia a poslední, přímý zápis do registru umožní radiofrekvenční přerušení.

3.2.5 Přerušení

Aplikace využívá 2 typy přerušení: Přerušení časovačů (vektory 1 a 3) a přerušení rádia (vektor 9). Obslužné procesy přerušení jsou definovány v souboru aplikace config.c. Pokud MCU zaznamená přerušení, MCU začne okamžitě vykonávat obslužný proces na adrese na kterou ukazuje vektor s tímto přerušením asociovaný. Pokud se neobjeví další přerušení vyšší priority, MCU servisní proces přerušení dokončí a pokračuje dále ve vykonávání kódu v místě, ze kterého do přerušení vstoupil.

Pro správnou funkci aplikace je nejdůležitější obsluha RF přerušení. Existují 3 typy RF přerušení již popsány v kapitole 2.2.5. Tyto principy umožňují aplikaci rychle reagovat na vzniklé situace (zpráva doručena/odeslána, dosažen maximální počet znovuodeslání). V případě odeslání paketu konstrukce programu zajišťuje, že nebudou do zásobníku TXFIFO přesunuta další data dříve, nežli dojde k některému z RF přerušení. Tím se zabrání případnému přetečení zásobníku a ztrátě dat.

Přerušení časovačů je využíváno v aplikaci využíváno ve dvou případech. V prvním případě jde o jednoduché bliknutí led diodou. Aby nemuselo být přesně definováno ve kterém místě programu se led dioda vypne, nebo aby čekání na uplynutí intervalu svícení led diody zbytečně nezdržovalo MCU je led dioda zapnuta, zároveň je odstartován i časovač. MCU pokračuje ve vykonávání dalších instrukcí a led dioda je vypnuta až ve chvíli kdy nastane příslušné přerušení časovače.

Druhý případ využití je obdobný. Časovač je použit k načasování libovolné události v horizontu několika vteřin. Maximální doba kterou je možné 16 bitovým časovačem odměřit je zhruba 50ms. Delší doby lze dosáhnout tak, že časovač je neustále spouštěn, při každém přerušení je

časovač vynulován a provede se inkrementace proměnné a až ve chvíli kdy tato proměnná dosáhne předem stanovené hodnoty, je provedena požadovaná část funkčního kódu.

3.2.6 Logika řízení

Funkce programu je následující. Po připojení základní desky ke zdroji napájení (USB kabel nebo baterie) dojde ke spuštění programu. Nejprve proběhne konfigurace tak jak bylo popsáno v kapitole 3.2.4. Poté program vstoupí do nekonečné smyčky. V tomto bodě se funkce liší podle toho zda jde o primární přijímač, nebo vysílač.

Primární vysílač (TX)

Aplikace pro primární vysílač je ovládána uživatelem pomocí třech digitálních vstupů, reprezentovanými třemi tlačítky. Po startu programu zařízení čeká na vstup z tlačítka. Každé tlačítko má 2 funkce.

- Tlačítko 1
  - Na stisk je odeslán 1 paket.
  - Podržení tlačítka po dobu delší než jedna vteřina vypne nebo zapne funkce Enhanced ShockBurst a odešle konfigurační paket do přijímače.
- Tlačítko 2
  - Na stisk je odesláno 256 paketů.
  - Podržení tlačítka po dobu delší než jedna vteřina přepíná datový tok mezi 2 Mbps (implicitní nastavení) a 1 Mbps a odešle konfigurační paket do přijímače.
- Tlačítko 3
  - Na stisk je odesláno 16384 paketů.

Pokud je zahájeno odesílání paketů, led diody, slouží k signalizaci stavu jeho doručení (led dioda 1) nebo ztráty (led dioda 2). Program využívá příslušných přerušení pro řízení svícení led diod.

Při stisku tlačítka je spuštěn časovač nastavený na 1 ms. S každým doběhnutím časovače je inkrementována proměnná „count“ a následně je její hodnota porovnána s nastavenou hodnotou (hodnota 1000 dá interval 1 s). Pokud proměnná „count“ této hodnoty dosáhne, program provede akce potřebné k nastavení daného parametru. Nejprve vyšle konfigurační paket, jehož obsah určuje jaké nastavení provést v přijímači, a poté je provedena změna parametrů v samotném vysílači. Změna parametrů je signalizována opakovaným probliknutím všech led diod. Pokud je tlačítko uvolněno dříve, než mohl časovač tisíckrát doběhnout, je časovač zastaven a proměnná „count“ vynulována. A program provede základní akci příslušející danému tlačítku.

Případná ztráta konfiguračního paketu je ošetřena pouze pro případ komunikace s automatickým potvrzováním a to nastaveným počtem znovuodeslání paketu. Pokud je potvrzování vypnuto tak nemá vysílač žádnou zpětnou vazbu od přijímače a považuje paket za doručený. Pokud se ani v jednom z režimů nepodaří paket odeslat, vysílač změní své nastavení, kdyby to neudělal a změna parametrů by závisela na stavu doručení konfiguračního paketu, mohlo by dojít ke kolizní situaci, kdy se vysílač v režimu s potvrzováním snaží nastavit přijímač v režimu bez potvrzování. Taková vzájemná komunikace však není možná, pakety by se neustále ztrácely a rádiové moduly by nemohly komunikovat vůbec. V případě ztráty konfiguračního paketu je proto třeba vysílač manuálně přepnou zpět do původního režimu a pokusit se o vzdálenou konfiguraci znova.

Program při požadavku na změnu nastavení kontroluje firmou Nordic předdefinovanými funkcemi aktuální konfiguraci a požadovaný parametr změní. Příkaz `if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)` kontroluje zda je komunikační potrubí otevřeno s automatickým potvrzováním či nikoli. Návratový typ funkce je unsigned char a vrací hodnotu 0x03 při stavu automatické potvrzování zapnuté a potrubí otevřené. Druhý příkaz `if(hal_nrf_get_datarate() == 0x00)` zjišťuje aktuální nastavení rychlosti datového toku. Funkce vrací hodnotu 0x00 pro datový tok 1 Mbps.

Primární přijímač (RX)

Aplikace pro primární přijímač rovněž využívá tlačítek a led diod k ovládání a signalizaci. Po spuštění je rádiový modul po provedení startovní inicializace okamžitě schopen přijímat pakety. Program reaguje na výskyt přerušení signalizující doručený paket a podle jeho obsahu provede patřičné akce. Pokud byl doručen konfigurační paket, provede program příslušnou změnu parametrů. Stav rádiového modulu je signalizován led diodami. Při základním nastavení, v klidovém stavu (nebul doručen nový paket) nesvítí žádná dioda.

- Enhanced ShockBurst vypnut – svítí led dioda 1.
- Datový tok snížen na 1 Mbps – svítí led dioda 2.
- Doručen nový datový paket – bliká led dioda 3.

Program počítá doručené datové pakety. Jejich počet, nebo hodnotu posledního doručeného paketu lze zobrazit v binárním kódu pomocí led diod na základní desce modulu. Program vstoupí do zobrazovacího po podržení tlačítka 1 (zobrazení počtu doručených paketů) nebo tlačítka 2 (zobrazení hodnoty posledního paketu) po dobu alespoň 1 vteřina (využití časovače jako v případě vysílače). V zobrazovacím režimu je přijímací režim vypnut změnou hodnoty RFCE bitu na 0. Program umožňuje zobrazit až 16 bitové binární číslo. Toto číslo je rozděleno po 8 bitech na vrchní a spodní byte. Nejprve je zobrazen spodní byte. Mezi bity v bytu se lze přesouvat pomocí tlačítek, tlačítko 3 zobrazí nejméně významné bity a tlačítko 1 nejvíce významné bity. Led dioda 3 zobrazuje vždy nejméně významný bit a dioda 1 nejvíce významný bit. Přepnutí na zobrazování horního bytu lze provést stiskem libovolného tlačítka po dobu jedné sekundy. Rozlišení zdali je právě zobrazován horní nebo dolní byte je zajištěno blikáním diody. U dolního bytu je to dioda 3, vpravo od nultého, nejméně významného bitu, pro horní byte je to dioda 1, vlevo od patnáctého, nejvíce významného bitu (obrázek č. 19). Program se vrátí do přijímacího režimu po souběžném stisknutí 1. a 3. tlačítka, to je možné ve kterékoli fázi zobrazování.

Obrázek č. 19 – zobrazení 16 bitového čísla.

Základem zobrazovací funkce je algoritmus převodu čísla do jeho binární interpretace. Do funkce vstoupí číslo, které má být zobrazeno, toto číslo je převedeno na binární a hodnoty

jednotlivých bitů jsou uloženy do binárního pole o šestnácti prvcích. Algoritmus převodu je následující:

```
for (j = 0; j < 16; j++)
{
   z = p % 2;
   p = p / 2;
   bin_pole[j] = z;
}
```

Proměnná „p“ je číslo která chceme zobrazit, to je v každém cyklu nejprve podrobeno operaci modulo 2 (zbytek po dělení dvěmi) a poté je vyděleno dvěma. Zbytek po dělení dvěmi (může nabývat hodnot pouze 0 nebo 1) se uloží do pole „bin_pole“ na příslušnou pozici. Proměnná „bin_pole“ je pak převzata další funkcí, která pomocí systému podmínek kontrolující jednotlivé pozice v poli rozsvěcuje příslušné led diody.

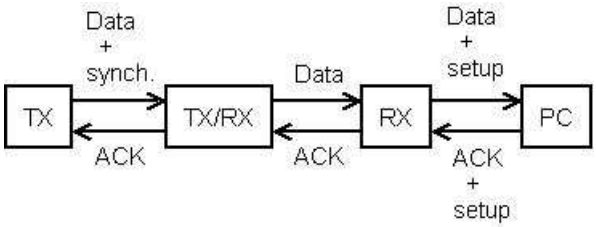
3.3 Komunikace mezi třemi body

3.3.1 Požadavky

Tato hlavní aplikace pro bezdrátovou body má za úkol zprostředkovat jednosměrnou komunikaci mezi 3 body v sérii. To znamená, že bod A vyšle zprávu, bod B ji zachytí a pošle ji do bodu C, který zprávu zpracuje. Stěžejní požadavek na aplikaci je možnost přepnout rádiový modul B bez nutnosti kompletního restartu MCU z režimu primární přijímač do režimu primární vysílač a zpět. Aplikace umožňuje otestovat komunikaci na chybovost přenosu, jednoduchým počítáním došlých paketů v koncovém bodě C. Informace o počtu doručených paketů je po zásahu uživatele (stisk tlačítka) odeslána do PC po USB. Tato informace může být dále porovnána s množstvím odeslaných paketů z bodu A. Toto lze provést pro různé nastavení přenosových parametrů rádiových modulů. Například pro přenos s 16, 8 bitovým CRC, nebo bez CRC, dále se zapnutým nebo vypnutým „retransmitem“ (znovuodeslání zprávy pokud nebyla doručena), popřípadě lze nastavit parametr počet těchto znovupřenosů.

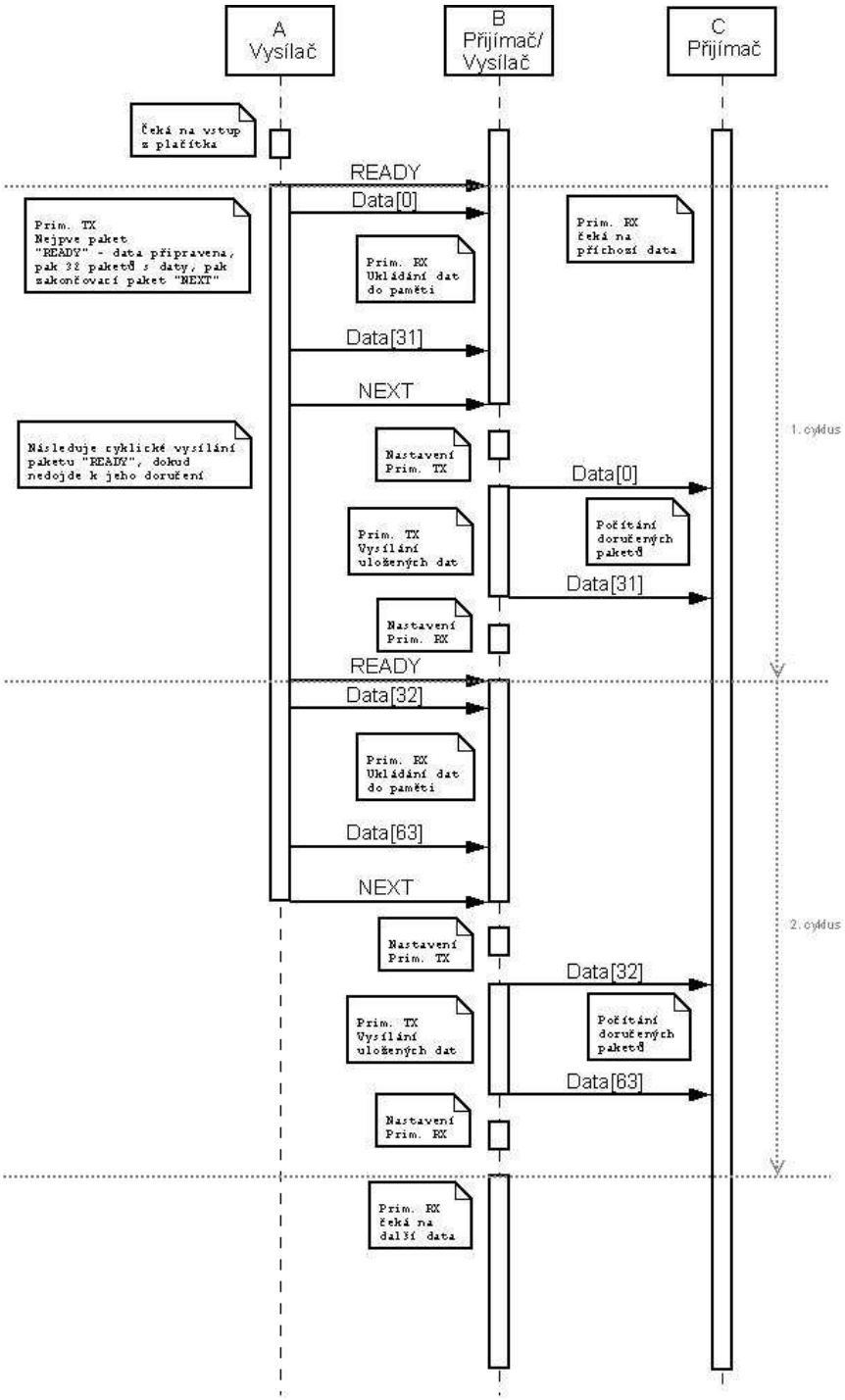
3.3.2 Návrh komunikačního systému

Podstata problematiky komunikace mezi třemi body vyžaduje vytvoření programu pro každý modul zvlášť, protože každý bude mít, co se logiky vykonávaného programu týče, rozdílnou funkci. Základní struktura programu se strukturou první úlohy. Po zapnutí rádia je nejprve nutná konfigurace jeho systémových registrů, které řídí chod rádiového modulu. Dále již následuje cyklicky se opakující část kódu, která řídí samotnou komunikaci. U komunikačního bodu A a C se v programu na určitých místech počítá se zásahem uživatele - řídící povely pomocí vstupů tlačítek, prostřední komunikační bod B pracuje samostatně. Principy komunikace jsou vysvětleny v předchozí kapitole (kapitola 3.2 Komunikace mezi dvěma body). V tomto případě přidání prostředního komunikačního bodu sebou nese problém synchronizace. Blokové schéma komunikace tří bodů a PC je na obrázku č. 20.



Obrázek č. 20 – Blokové schéma komunikace 3 bodů a PC v sérii.

Při návrhu synchronizačního algoritmu je potřeba zajistit co možná nejrychlejší spolupráci jednotlivých komunikačních bodů. S tím souvisí maximální velikost přímo adresovatelné interní RAM paměti, která je 128 bytů. Proto u prostředního komunikačního bodu B je nezbytné zajistit střídání režimu přijímání a vysílání. A to tak, aby byla paměť zaplněná přijatými daty vyprázdněna dřív, než dojde k jejímu přetečení. Aby bylo splněna podmínka, že všechna odesílaná data z bodu A budou zároveň i přijímána a aby byly minimalizovány časové prostoje při změnách komunikačních režimů, je třeba systém adekvátně navrhnout. Časový diagram spolupráce tří komunikačních bodů je znázorněn na obr. č. 21.



Obrázek č. 21 – Časový diagram komunikace mezi třemi komunikačními body.

V diagramu je znázorněna navržená struktura komunikace. V tomto případě je odesláno 64 paketů, což pro střední komunikační bod B znamená provedení dvou cyklů střídání režimu primární vysílač, primární přijímač.

Komunikaci zahajuje bod A, bod B a C jsou po spuštění v režimu přijímání a čekají na první paket. Po zmáčknutí tlačítka na základní desce začne bod A vysílat paket, který bude v bodu B identifikován jako paket s pokynem „READY“. Toto je synchronizační paket, který slouží k upozornění středního komunikačního bodu na blížící se balík dat. Bod A vysílá paket „READY“ tak

dlouho, dokud není bodem B potvrzeno jeho přijetí. Poté následuje kontinuální odeslání 32 paketů. Úspěšně přijaté pakety bodem B, jsou uloženy do paměti. Za těmito datovými pakety následuje opět synchronizační paket, který je v bodě B identifikován jako „NEXT“. Tento balík upozorní střední bod na konec vysílání prvního balíku dat. Přijetí a potvrzení paketu „NEXT“ přepne prostřední bod do primárního vysílacího režimu, bod A zatím cyklicky „naprázdno“ vysílá paket „READY“. Poté co v bodě B došlo k rekonfiguraci systému na primární vysílač, jsou data z paměti odeslána směrem k bodu C, který do této chvíle čekal na svůj první paket. V případě doručení paketu, bod C inkrementuje čítač došlých paketů a doručená data může dále zpracovat. Po odeslání všech dat z paměti je bod B přepnut opět do přijímacího režimu a čeká na další paket, kterým bude paket „READY“ oznamující další balík dat.

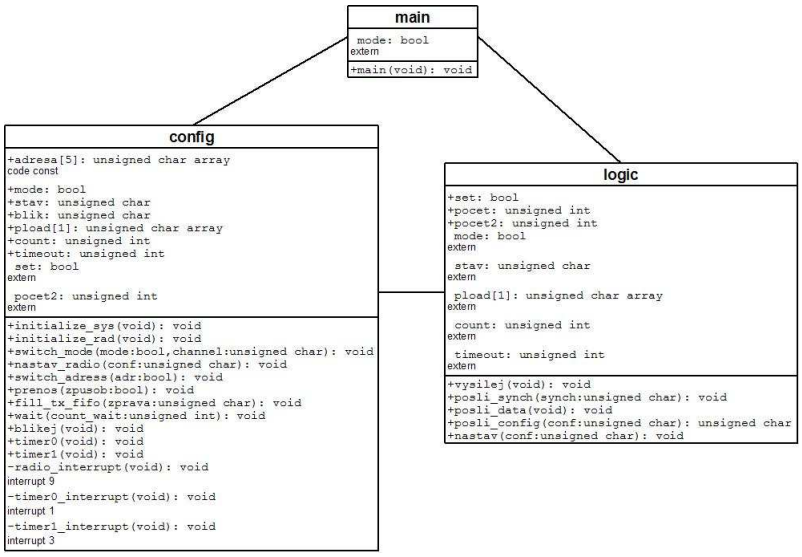
Došlá data je možné, ve chvíli kdy přijímač čeká na novou várku paketů z prostředního bodu, odeslat do PC po USB rozhraní. Pro test aplikace toho ovšem nebude využito a do PC bude odesílán pouze dvoubytový údaj o počtu doručených paketů.

Aby nedošlo ke kolizi komunikace, tzn. aby bod C nepovažoval pakety od bodu A za svá data, je třeba komunikační kanály od sebe rozlišit. Existují 2 způsoby, jak to udělat. Jednak je možné nastavit jinou frekvenci pro komunikaci bodu B s bodem C, ale daleko vhodnější řešení je pakety určené pro bod C jinak adresovat. Přijímač v tomto případě přijme všechny pakety, ale pakety s adresou jinou, než jakou přijímač očekává jsou ihned zahozeny.

3.3.3 Struktura aplikace

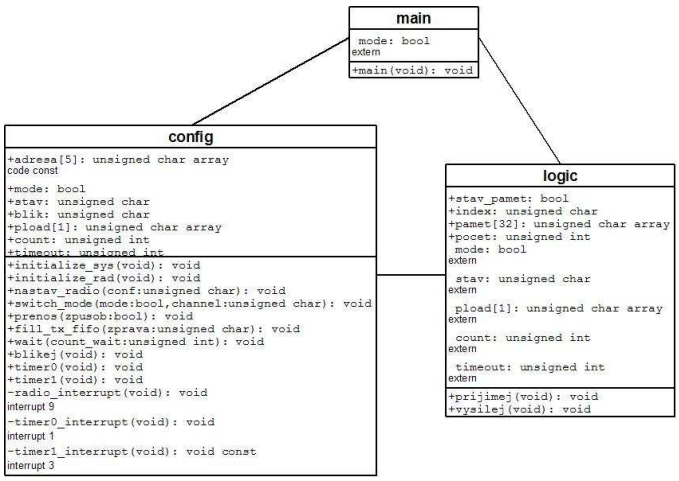
Struktura projektu hlavní aplikace je totožná se strukturou aplikace komunikace mezi dvěma body (kapitola 3.2.3). Stejně jako u předchozí aplikace, i zde je zvlášť pro každý komunikační bod vytvořen jeden program. Jednotlivé programy mají nastavení registrů stejné jako aplikace komunikace dvou bodů, s tím rozdílem, že prostřední komunikační bod v čase mění své nastavení z primárního přijímače na vysílač tak, jak je to naznačeno v diagramu na obrázku č. 21.C soubory projektu, vztahy mezi nimi, jejich funkce a proměnné jsou zobrazeny v diagramech na obrázcích č. 22, 23 a 24

Komunikační bod A - Primární vysílač (TX):



Obrázek č. 22 – Struktura projektu pro komunikační bod A.

Komunikační bod B – Přijímač/Vysílač (RXTX):



Obrázek č. 23 – Struktura projektu pro komunikační bod B.

Komunikační bod C – Primární přijímač (RX):

Koncový bod jako jediný využívá připojení k PC pomocí USB. Proto projekt pro tento komunikační bod obsahuje soubor usbfces.c. V tomto projektovém souboru se nalézají definice funkcí a přerušení spojených s USB komunikací. Více o nastavení USB řadiče v kapitole č. 3.3.4 – Konfigurace spojení s PC.

Obrázek č. 24 – Struktura projektu pro komunikační bod C.

Programové soubory v každém projektu spolu spolupracují, předávají si proměnné, využívají své funkce atd. V diagramech znaménko „+“ před proměnnou, znamená že proměnná je globální. Je možné ji využít i v jiném .c souboru přidáním výrazu „extern“ před definici proměnné. Znaménko „+“ před uvedenými funkcemi znamená, že jsou jejich prototypy uvedeny v hlavičkových souborech a tudíž je lze využívat i mimo soubor ve kterém jsou definovány. # symbolizuje proměnné nebo funkce typu „static“. Dále se v souboru „usbces“ vyskytuje proměnná typu „volatile“. Toto označení umožňuje manipulovat s proměnnou ve vnějším procesu, například při přerušení.

3.3.4 Propojení s PC a vizualizace dat

Úloha komunikace pro 3 body využívá PC k vizualizaci informací o přenosu dat. Spojení PC a rádiového modulu nRF24LU1 je zprostředkováno pomocí USB. Implementovaný USB řadič je třeba nejprve nakonfigurovat. K tomu jsou využity přiložené programové šablony firmy Nordic. Nejdůležitější je hlavičkový soubor hal_usb.h, ve kterém se nalézají prototypy funkcí, zprostředkovávajících nastavení a chod USB komunikace. Definice těchto funkcí se nachází v souboru hal_usb.c, který je třeba do projekt rovněž importovat. Vzhledem k tomu, že registry pro kontrolu USB řadiče rádiového modulu se nacházejí v prostoru „XDATA“, což je fixní nepřímo adresovatelná část paměti rádiového modulu, je třeba do projektu dodat soubor STARTUP.A51. Tento soubor je součástí kompilátoru C51 a mimo jiné umožňuje práci s XDATA registry.

Projektový soubor usbfces.c implementuje funkce USB rozhraní a „callback“ funkce. Ihned po startu programu proběhne funkce initialize_usb a provede základní inicializaci a založí koncové body pro odesílání dat. Obsahuje příkazy:

```
hal_usb_init(true,         callback_device_req,         callback_suspend,
callback_resume, callback_reset);
hal_usb_endpoint_config(0x81, 1, callback_endpoint1_in);
hal_usb_endpoint_config(0x82, 2, callback_endpoint2_in);
```

Spolu s inicializací USB vznikají automaticky 2 servisní koncové body, jeden pro příjem, druhý pro odesílání konfiguračních a stavových zpráv. Příkaz usb_init (definován v hal_usb.c) nastaví odkazy na reentrantní funkce do kterých program v průběhu USB komunikace vstupuje. Hal_usb_endpoint_config založí koncový bod pro komunikaci. První parametr určuje směr komunikace a číslo koncového bodu. 0x81 znamená komunikaci směrem k hostitelskému USB řadiči a přiděluje koncovému bodu číslo 1. Pro příjem dat od hostitele by byl tento parametr 0x01. Druhý parametr určuje počet bytů k odeslání, nebo příjmu a třetí dává odkaz na další reentrantní funkci.

V dalším kroku je třeba aby program vyčkal na provedení této konfigurace a rozpoznání zařízení počítačem, což může trvat i několik vteřin. Smyčka while proto cyklicky provádí kontrolu stavu USB řadiče příkazem:

```
hal_usb_state_t hal_usb_get_state();
```

Jakmile příkaz vrátí stav „CONFIGURED“ je smyčka ukončena a program může pokračovat. Tento stav je signalizován opakovaným blikáním všech led diod na základní desce a charakteristickým zvukem v operačního systému Windows XP.

Pro samotné odesílání paketu je použita funce:

```
hal_usb_send_data(0x81, in_data, 1);
```

Parametry jsou: koncový bod, data pro odeslání a počet bytů zprávy. Použití tohoto příkazu je podmíněno stavem binární proměnné „endpoint1“. Proměnná je po odeslání paketu vynulována a dokud ji příslušná reentrantní funkce daného koncového bodu nenastaví opět na hodnotu 1, nemůže být odeslán další paket.

V souboru usbfces jsou definovány tyto reentrantní funkce:

```
static   hal_usb_dev_req_resp_t   callback_device_req(hal_usb_device_req*
req, uint8_t** data_ptr, uint16_t* size) reentrant
```

Funkce zpětného volání která je použita když je přijat dotaz od hostitele.

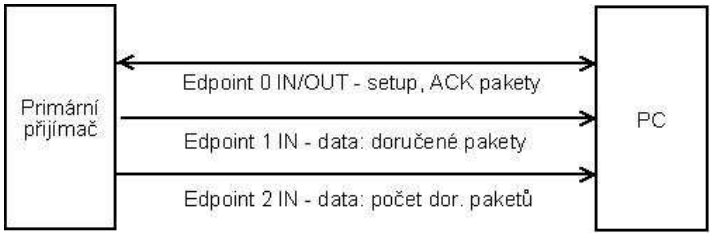
Funkce odpovídá zprávami:

- STALL - komunikace zastavena
- NAK - no acknoledgement
- ACK - acknoledge (při přijmu dat)
- NO_RESPONSE - žádná odpověď
- DATA - data připravena
- EMPTY_RESPONSE - pošli prázdnou odpověď

```
static void callback_suspend(uint8_t allow_remote_wu) reentrant
static void callback_resume() reentrant
static void callback_reset() reentrant
```

Tyto funkce jsou volány pokud se na USB rozhraní objeví požadavek na pozastavení, obnovení nebo resetování komunikace. Tyto funkce jsou volány pokud nastane přerušení daného koncového bodu.

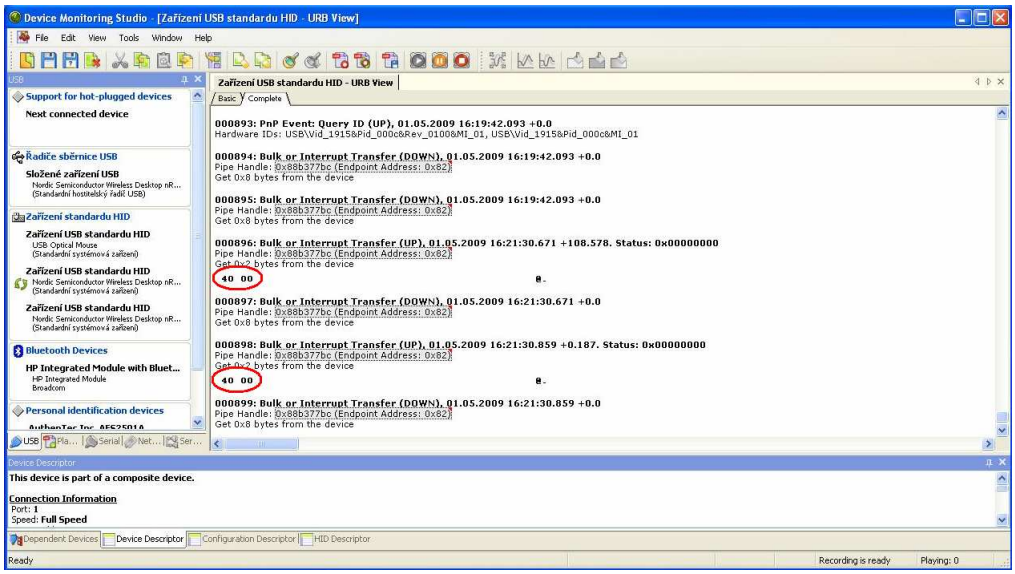
```
static  uint8_t  callback_endpoint1_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant
static  uint8_t  callback_endpoint2_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant
```



Obrázek č. 25 – Schéma USB komunikace.

Prototypy vytvořených funkcí jsou umístěny v hlavičkovém soubory usbfces.h. Ten umožňuje použití těchto funkcí i v jiných projektových souborech.

USB komunikace je na straně PC monitorována zkušební verzí programu Device Monitoring Studio. Program sleduje tok dat mezi PC a externími USB zařízeními. Datový obsah odesílaných paketů je uváděn v hexadecimálním kódu (na obrázku č. 26 zakroužkováno červeně).



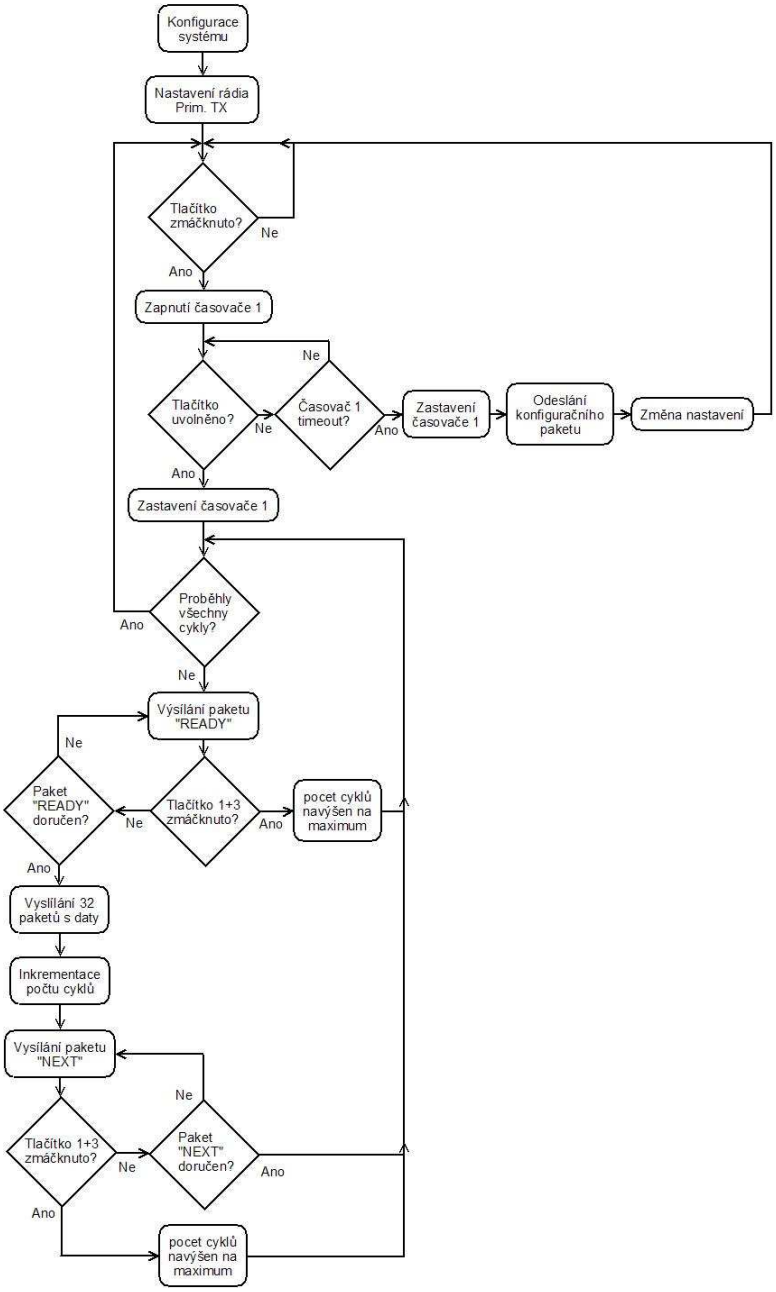
Obrázek č. 26 Device Monitoring Studio.

3.3.5 Logika řízení

Komunikační bod B, primární vysílač (TX)

Ovládání programu je totožné jako u aplikace pro 2 komunikační body. Třemi tlačítky na základní desce si uživatel vybere, kolik dat chce odeslat. Na výběr je 1 byte, 256 bytů, a 65535 bytů.

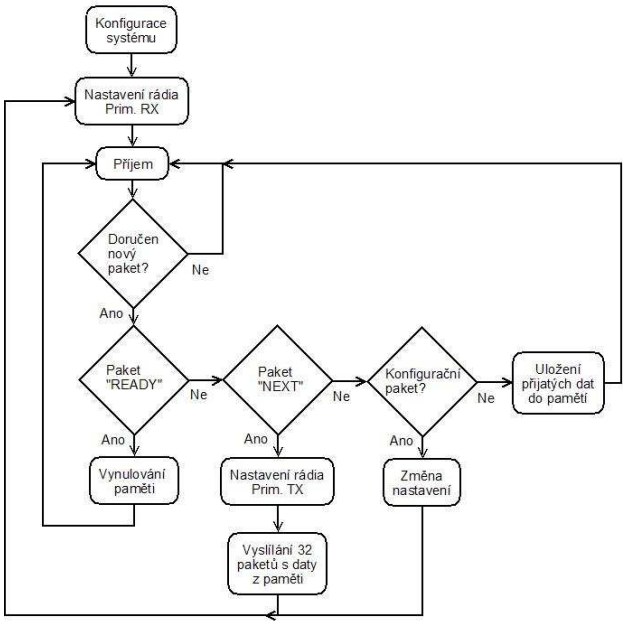
Pokud uživatel podrží tlačítko 1 nebo 2, jsou pozměněny parametry komunikace. Opět jako u jednodušší aplikace, lze zapnout, nebo vypnout automatické potvrzování paketů (podržení tlačítka 1), nebo měnit datový tok ze 2 na 1 Mbps (podržení tlačítka 2). Konfigurační paket je ale v případě této komunikace odeslán nejen do prostředního komunikačního bodu B, ale i do koncového přijímacího bodu C. Vysílač nejprve odešle paket s adresou pro bod B a pak s adresou pro bod C. Přesný chod programu je znázorněn v blokovém diagramu na obrázku č. 27.



Obrázek č. 27 – Blokový diagram chování programu pro komunikační bod A.

Komunikační bod B, vysílač/přijímač

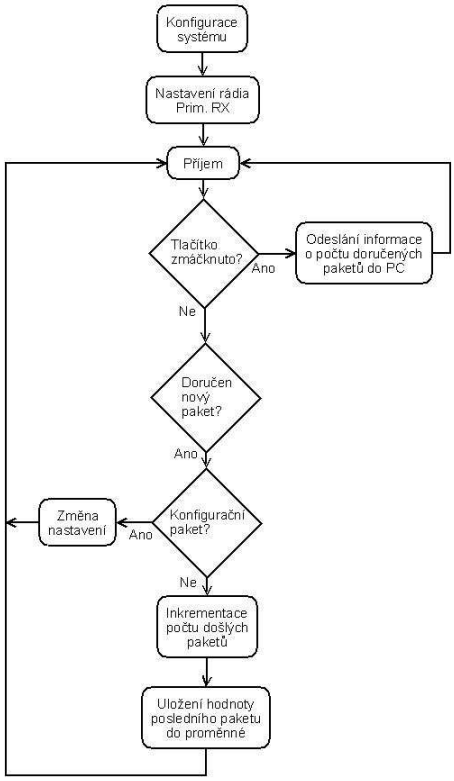
Pro střední komunikační bod je použit malý USB dongle, což neumožňuje uživateli jakkoli do běhu programu zasahovat. Program je z tohoto důvodu naprosto autonomní. Program na základě obsahu přijatého paketu rozhoduje, jaká akce bude dále provedena. Princip byl nastíněn v kapitole č. 3.3.2 a znázorněn na obrázku č. X. Program nereaguje jen na synchronizační pakety „READY“ a „NEXT“, ale také na konfigurační paket. Pokud je detekován, rádio provede příslušnou změnu nastavení. Chod programu pro střední komunikační bod je znázorněn na obrázku č. 28.



Obrázek č. 28 – Blokový diagram chování programu pro komunikační bod B.

Komunikační bod C, primární přijímač

Program pro koncový komunikační bod C je totožný s programem přijímacího bodu v aplikaci komunikace dvou bodů, s tím rozdílem, že v tomto případě má rádiový modul nastavenou rozdílnou adresu pro příjem paketů. Program setrvává neustále v přijímacím režimu. Pokud je v průběhu chodu programu zmáčknuto tlačítko 1 na základní desce, je do PC odeslána informace o počtu do této chvíle doručených paketů. Blokovým diagram na obrázku č. 29 chod programu znázorňuje.



Obrázek č. 29 – Blokový diagram chování programu pro komunikační bod C.

3.4 Vyhodnocení míry chybovosti bezdrátového přenosu

U obou úloh bezdrátové komunikace lze navolit několik parametrů, které podstatně ovlivní probíhající komunikaci a to převážně ze dvou hledisek. První je celková rychlost přenosu paketů ze zdrojového bodu, do koncového bodu. Druhé hledisko je míra chybovosti paketů (PER) což je poměr mezi nedoručených paketů k počtu odeslaných paketů. Tyto vlastnosti komunikace nebudou ovlivněny jen nastavenými parametry, ale také vzájemnou vzdáleností rádiových modulů. Datasheet k vývojovému kitu nRF24LU1 definuje maximální dosah jako vzdálenost při které dosáhne přijímač limitu citlivosti, což odpovídá míře chybovosti bitů 1/1000. Míra citlivosti bitů (BER) je poměr mezi chybnými bity a všemi bity v paketu. Na rozdíl od počtu nedoručených paketů (ve skutečnosti se počítávají doručené pakety) nelze počet chybných bitů pomocí rádiových modulů nRF24LU1 přímo měřit. Koeficient BER však lze z PER vypočítat a to pomocí následujícího vztahu:

$$BER = 1 - \sqrt[n]{(1 - PER)}$$

Písmeno „n“ ve vzorci znamená počet bitů na jeden paket. V případě obou aplikací se paket skládá z 81 bitů pro přenos s automatickým potvrzováním. 1 úvodní byte, 5 bytů adresa, 9 bitů kontrolní pole paketu, 1 byte data a2 byty CRC na konci paketu.

Způsob jakým rádiový modul se zapnutým Enhanced ShockBurst, zjišťuje správnost doručení paketů na straně přijímače je následující:

Rádiový modul neustále kontroluje, zda-li nebyl doručen paket s platnou adresou, v případe neplatné adresy je paket ihned zahozen. Pokud je adresa platná, projde paket CRC kontrolou. Pokud CRC kontrola neprokáže chybu, je paket podroben PID kontrole. V případě shodného PID s předchozím paketem je následně srovnáno CRC předchozího paketu s nově příchozím. Shoda v obou kontrolách zároveň znamená doručení kopie předchozího paket a paket je zahozen. V opačném případě je paket odeslán do RX FIFO a rádio odešle vysílači potvrzení o doručení. Tímto postupem je zaručeno, že každý paket, který MCU z RX FIFO vyzvedne je nový, unikátní a neobsahuje chybu. Takové pakety lze počítat jako pakety doručené.

Na straně vysílače je počítán skutečný počet odeslaných paketů. Je sice vygenerován neměnný počet paketů k odeslání, ale pokud vysílač neobdrží ACK paket, opakuje odesílání paketu, tolikrát kolikrát má nastaveno (nastaveno na 3x), pak paket zahodí. Tyto přenosy „navíc“ je nutné zahrnout do přese kalkulace PER.

3.4.1 Vyhodnocení chybovosti pro komunikaci 2 bodů

Test spočívá ve zvyšování vzdálenosti mezi rádiovými moduly a počítání doručených paketů na straně přijímače a měření času přenosu. Pro každou vzdálenost je odesláno 16384 paketů s 1 bytem dat. Pouze 1 byte dat byl zvolen záměrně (maximum může být 32 bytů na paket), aby bylo odesláno co nejvíce paketů za co nejkratší dobu. Z počtu doručených paketů lze vypočítat míru chybovosti paketů a následně míru chybovosti bitů. Test se provádí pro s parametry automatické potvrzování zapnuto a vypnuto a také pro datové toky 1 a 2 Mbps. Výsledky jsou uvedeny v tabulce č. 1.

	2 Mbps					1 Mbps				
l [cm]	***Odeslané pakety***	***Ztracené pakety***	***Čas [s]***	***PER***	***BER***	***Odeslané pakety***	***Ztracené pakety***	***Čas [s]***	***PER***	***BER***
50	16385	1	6,5	0,000061	0,000001	16384	0	8	0,000000	0,000000
100	16394	10	6,5	0,000610	0,000008	16390	6	8	0,000366	0,000005
150	18083	1711	7,5	0,094619	0,001226	16628	250	8	0,015035	0,000187
200	20878	4606	8,5	0,220615	0,003072	16894	518	8	0,030662	0,000384
250	39911	25036	17	0,627296	0,012111	18613	2597	9	0,139526	0,001853
300	58494	47998	23,5	0,820563	0,020986	32097	16153	15	0,503256	0,008601
350	64679	61510	28	0,951004	0,036550	44289	29210	21	0,659532	0,013214
400	65002	64352	29	0,990000	0,055268	63592	57513	30	0,904406	0,028567

Tabulka č. 1 – Test komunikace 2 bodů s ACK.

Hodnotu PER*100 lze také interpretovat jako procentuální chybovost paketů, stejně tak BER*100 jako procentuální chybovost přenosu bitů.

V případě přenosu s automatickým potvrzováním není znám přesný počet odeslaných paketů, protože nedoručení paketu znamená jeho opětovné odeslání, to je prováděno dokud není dosaženo maximálního počtu znovuodeslání paketu, který je u všech testů nastaven na hodnotu 3. Počet odeslaných paketů pro test s ACK se tedy pohybuje v rozmezí 16384 – 65536. To a fakt že Enhanced ShockBurst neustále přepíná rádiový modul mezi režimem TX a RX (vysílání a příjem ACK paketu) má za následek markantní zpomalení komunikace oproti komunikaci bez ACK. Nižší datový tok má za následek vyšší citlivost přijímače, což má za následek zvýšení počtu přijatých paketů na stejné vzdálenosti, za cenu zvýšení času.

Informace o počtu doručených paketů je odesílána z přijímače po USB do PC. K správnému provedení výpočtu BER je třeba znát přesný počet vyslaných paketů (je třeba uvažovat i znovuodesílané pakety při případné ztrátě), to zajišťuje přijímač pomocí příkazu `hal_nrf_get_transmit_attempts()`, který vrací počet pokusů o znovuodeslání paketu. Naměřená data potvrdily předpoklady. Pro komunikaci s ACK dosahuje komunikace relativně malé chybovosti, s nevýhodou vyšších komunikačních časů. Časové ztráty vzniklé automatickým potvrzováním lze redukovat zvýšením počtu bytů dat na 1 paket. Snížení datového toku znamená lehké zvýšení citlivosti a počtu doručených paketů na úkor ztráty přenosové rychlosti.

S výsledků vyplývá, že hranice citlivost 1/1000 BER se nachází ve vzdálenosti 140 cm, zařízení s nižším datovým tokem 1 Mbps má dosah podstatně vyšší, pohybuje se okolo 2,2m. Pro srovnání s aplikací komunikace 3 bodů je uveden graf závislosti míry bitové chybovosti na vzdálenosti vysílacích bodů.

3.4.2 Vyhodnocení chybovosti pro komunikaci 3 bodů

Druhý test probíhá obdobně, Tři komunikační body jsou rozprostřeny v sérii na postupně na vzdálenostech 0,5 a 4 metry a z přijímače je odesláno 16383 paketů. V tomto případě je sledována komunikace jako černá skříňka, nejsou podstatné procesy ve středním komunikačním bodu. Jsou měřeny pouze pakety odvysílané bodem A a pakety přijaté bodem C.

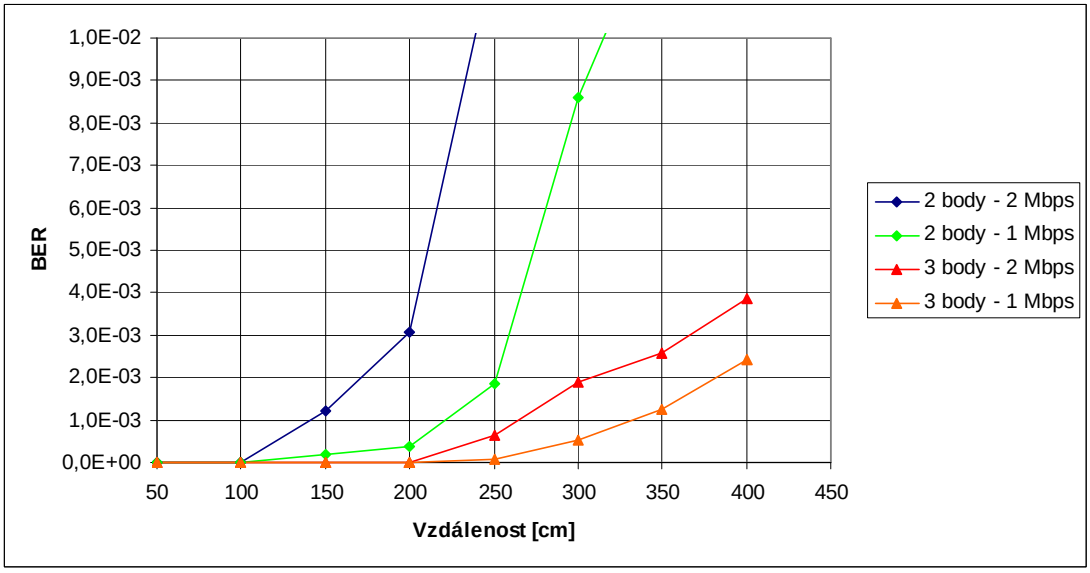
Na tomto příkladě je demonstrovány výhody a nevýhody přidání středního komunikačního bodu. Nesporná výhoda je zvýšení spolehlivosti komunikace. Teoreticky se snížením přenosové vzdálenosti na polovinu, by se měla zvýšit spolehlivost na dvojnásobek. To se však nestane, protože

všechna data jsou posílána na dvakrát, tzn. Dochází k vyššímu počtu chyb. S tím spojená nevýhoda je však značné snížení přenosové rychlosti. Tento pokles je způsoben transferem dat přes střední komunikační bod, kde ono podstatné zpomalení vzniká. Výsledky jsou uvedeny v tabulce č. 2.

	2 Mbps					1 Mbps				
l [cm]	***Odeslané pakety***	***Ztracené pakety***	***Čas [s]***	***PER***	***BER***	***Odeslané pakety***	***Ztracené pakety***	***Čas [s]***	***PER***	***BER***
50	16384	0	36	0,000000	0,000000	16384	0	40	0,000000	0,000000
100	16391	7	36	0,000427	0,000005	16384	0	40	0,000000	0,000000
150	16389	5	36	0,000305	0,000004	16386	2	40	0,000122	0,000002
200	16398	14	36	0,000854	0,000011	16390	6	40	0,000366	0,000005
250	17195	873	37,5	0,050771	0,000643	16481	105	41	0,006371	0,000079
300	18911	2676	39	0,141505	0,001882	17063	716	42	0,041962	0,000529
350	19969	3770	40,5	0,188793	0,002580	17921	1707	43,5	0,095251	0,001235
400	21419	5756	41	0,268733	0,003856	19580	3506	44	0,179060	0,002433

Tabulka č. 2 – Test komunikace 3 bodů s ACK.

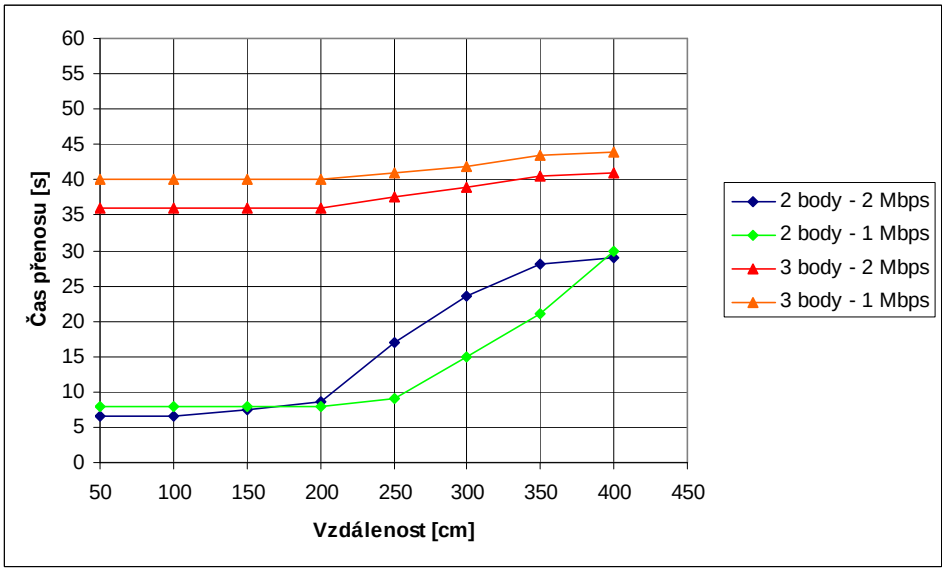
Z naměřených výsledků vyplývá, že použití tří a více komunikačních bodů tohoto typu je vhodné pro vzdálenosti které přesahují dosah jednoho zařízení. Dosah série všech komunikačních bodů jako celku je v tomto případě větší. Graf závislosti BER na vzdálenosti ukazuje, že hranice citlivosti 1/1000 BER byla překročena pro soubor všech komunikačních bodů až ve vzdálenosti 2,5m pro hustotu datového toku 2 Mbps a pro datový tok 1 Mbps se dosah komunikačního řetězce pohybuje nad hranicí 3 m.



Graf č. 1 – Závislost BER na vzdálenosti krajních komunikačních bodů.

3.4.3 Časování přenosu

Celkové doby potřebné na přenos 16384 paketů s nastaveným automatickým potvrzováním a trojnásobným retransmitem nepřesáhly v žádném z případů 45s. Graf č. 2 zobrazuje srovnání časů přenosu testovaných úloh s různě nastaveným datovým tokem.



Graf č. 2 – Závislost času přenosu na vzdálenosti krajních komunikačních bodů.

Z grafu a tabulky je možno usoudit, že rostoucí míra chybovosti má vliv na čas odesílání. Pokud je paket vyhodnocen jako chybný, ACK paket není odeslán a TX modul čeká 250 µs a pak odešle původní paket znova. Tento faktor hraje hlavní roli v nárůstu času. Srovnání komunikace 2 bodů s různým nastavením datového toku přináší zajímavý poznatek. Papírově pomalejší 1 Mbps komunikace (zelená) dosahuje od 1,75 m vzdálenosti podstatně lepších časů než komunikace 2 Mbps (modrá). Časy se srovnají na 4 metrech. Na této vzdálenosti ztrácí komunikace 2 body, 2 Mbps až 99% všech odeslaných paketů, to znamená že čas přenosu již růst nemůže.

Dobu potřebnou k vykonání 1 Enhanced ShockBurst cyklu lze vypočíst podle vztahu:

$$T_{ESB} = T_{UT} + 2 \cdot T_{s\tan dby} + T_P + T_{PACK} + T_{IRQ}$$

T_{UT} znamená „upload time“ a je to bitová délka dat dělená rychlostí SPI rozhraní. $T_{standby}$ je doba potřebná ke změně na TX nebo RX režim (130 µs). T_P je doba přenosu dat a vypočte se jako délka paketu dělená rychlostí datového toku. T_{PACK} je doba přenosu ACK paketu a T_{IRQ} je doba potřebná k obsluze přerušení, což je pro 2 Mbps 6 µs a pro 1 Mbps 8,2 µs.

Pro komunikaci 2 bodů lze tímto způsobem spočítat teoretickou dobu potřebnou k odeslání daného počtu paketů. Tento vzorec ovšem nezahrnuje časy potřebné k vykonání dalších částí kódu programu, obsluhy jiných přerušení a především dobu kterou rádio čeká na potvrzovací paket (0 až nastavených 250 µs). Výpočet je dále naprosto nevhodný pro odhad doby přenosu pro úlohu komunikace tří bodů. Aplikace odesílá synchronizační pakety navíc a díky prostojům daných změnou režimu středního bodu je pro tuto aplikaci doba přenosu neodhadnutelná. V tabulce č. 3 jsou proto uvedeny odhady dob jen pro úlohu komunikace dvou bodů.

	2 Mbps			1 Mbps		
l [cm]	***Odeslané pakety***	***Změřená doba [s]***	***Vypočtená doba [s]***	***Odeslané pakety***	***Změřená doba [s]***	***Vypočtená doba [s]***
50	16385	6,5	5,5	16384	8	6,9
100	16394	6,5	5,6	16390	8	6,9
150	18083	7,5	6,2	16628	8	7,0
200	20878	8,5	7,2	16894	8	7,1
250	39911	17	13,7	18613	9	7,9
300	58494	23,5	20,1	32097	15	13,6
350	64679	28	22,2	44289	21	18,7
400	65002	29	22,4	63592	30	26,9

Tabulka č. 3 – Odhad dob přenosu pro komunikaci 2 bodů.

Závěrem lze tedy konstatovat, že zvýšení datového toku nemá na celkovou rychlost komunikace příliš zásadní vliv, ale podstatně ovlivní citlivost přijímače, což se na větších vzdálenostech projeví zvýšenou mírou chybovosti. Pro nižší vzdálenosti je datový tok 2 Mbps výhodnější. Pro vzdálenosti v rozmezí 1,5 – 2,3 m lepší zvolit datový tok 1 Mbps.

Pro přenos mezi dvěma komunikačními body bez ztrát je nejvyšší celková teoretická přenosová rychlost s datovým tokem 2 Mbps a zapnutým automatickým potvrzováním paketů a velikostí paketu 81 bitů (nastavení paketu: 5 bytů adresa, 1 byte užitečných dat, 2 byty CRC) 31 kB/s. V testu změřená rychlost se pro bezztrátový přenos s výše popsaným nastavením pohybuje okolo 26 kB/s. Pokud se zvýší poměr užitečných dat ke zbytku paketu může se teoretická přenosová rychlost vyšplhat až na 82 kB/s (nastavení paketu: 3 byty adresa, 32 bytů užitečných dat a 1 byte CRC).

Pro sériový přenos tří bodů nelze celkovou rychlost teoreticky specifikovat, protože hodně záleží na rychlosti s jakou je schopen střední komunikační bod data předávat dále. V testu se celková naměřená přenosová rychlost komunikačního řetězce s datovým tokem 2 Mbps a 81bitovým paketem pohybovala za ideálních podmínek (nulové ztráty) okolo 4,3 kB/s. Pokud pro aplikaci není potřebná vysoká rychlost přenosu a počítá se s nasazením ve větších vzdálenostech, nebo v prostředí kde by mohlo dojít ke stínění přímé komunikace, je vhodnější využít tří v sérii zapojených komunikačních bodů.

4 Závěr

Bakalářská práce shrnuje poznatky o problematice bezdrátové komunikace a především o praktické aplikaci této problematiky. Hlavní přínos práce je ve zdokumentování návrhu, realizace a testování původních úloh vytvořených pro vývojový kit nRF24LU1 firmy Nordic Semiconductors. Srovnání úlohy pro přenos informací mezi dvěma komunikačními body s úlohou bezdrátové komunikace tří bodů zapojených v sérii, přineslo zajímavé výsledky zejména z hlediska nahlížení na rychlost přenosu, chybovost komunikace a vlivu vzdálenosti na tyto parametry, což může být nápomocno při návrhu jiných aplikací pro rádiové moduly firmy Nordic. Tyto úlohy, jako praktické příklady využití vývojového kitu, informace a postupy návrhu těchto úloh pro bezdrátovou komunikaci, které práce přináší, jsou tedy použitelné pro další vývoj podobných aplikací, nebo mohou být využity ke studiu nebo výuce dané problematiky.

Práce splňuje předsevzaté cíle. Na základě získaných teoretických poznatků se podařilo vytvořit 2 funkční úlohy bezdrátové komunikace. Úloha komunikace 2 bodů a úloha sériové komunikace 3 bodů. Tyto úlohy byly následně otestovány s využitím PC jako vizualizačního nástroje, kdy bylo zřízeno spojení přes USB rozhraní a na PC běžící program monitoroval tok dat na příslušném USB portu. Vzájemné srovnání aplikací prokázalo teoretické předpoklady a potvrdilo výhodnost využití sériového propojení bezdrátových komunikačních bodů při přenosu na vzdálenosti mimo dosah jednoho samostatného rádiového modulu. Dále se podařilo určit efektivní dosah pro rádiový modul nRF24LU1 v závislosti na nastaveném datovém toku a experimentálně byla potvrzena výpočtem stanovená přenosová rychlost zařízení a bil určen vliv velikosti jednoho paketu na přenosovou rychlost .

Pro případné pokračování tohoto projektu by bylo vhodné rozvinout spolupráci vývojového kitu s PC, především možnost plné kontroly, zadávání parametrů, odesílání zpráv. To vše by mělo vést k aplikaci rádiových modulů pro konkrétní využití v praxi například nasazení HID zařízení.

5 Seznam použité literatury

[1]. NEVŘIVA, Pavel. Analýza signálů a soustav. 1. vyd. Praha : BEN - technická literatura, 2000. 670 s. ISBN 80-7300-004-0.

[2]. HAC, Anna. Wireless sensor netvork design. USA : John Wiley & Sons, 2004. 408 s. ISBN 0-470-86736-1.

[3]. *History of radio* [online]. c2002 , last modified on 1 February 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/History_of_radio>.

[4]. *Wireless* [online]. c2002 , last modified on 30 April 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Wireless_communication>.

[5]. *Cyclic redundancy check* [online]. c2002 , modified on 1 May 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Cyclic_redundancy_check>.

[6]. *Computation of CRC* [online]. c2002 , last modified on 19 April 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Computation_of_CRC>.

[7]. *Cryptography* [online]. c2002 , last modified on 29 April 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Cryptography>.

[8]. *Advanced Encryption Standard* [online]. c2002 , last modified on 29 April 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Advanced_Encryption_Standard>.

[9]. *nRF24LU1*: Product Specification v1.1 [online]. Versin 1.1 c 2002, Last revision November 2008 [cit. 2009-05-01]. Dostupný z WWW: <http://www.nordicsemi.com/files/Product/data_sheet/Product_Specification_nRF24LU1_v1_1.pdf>.

[10]. *nRF24LU1 Development Kit* : User Guide v1.0 [online]. Version 1.0. c2002 , Last revision September 2007 [cit. 2009-05-01]. Dostupný z WWW: <http://www.nordicsemi.com/files/Product/development_tools/nRF24LU1DevelopmentKitUserGuide.pdf>.

[11]. *nRF24LU1 Reference Design* : User Guide v1.0 [online]. Version 1.0. c2002 , Last revision September 2007 [cit. 2009-05-01]. Dostupný z WWW: <http://www.nordicsemi.com/files/Product/development_tools/nRF24LU1ReferenceDesignUserGuide.pdf>.

6 Seznam příloh

Přílohy lze nalézt na přiloženém DVD.

Přílohy

1 Zdrojové kódy

V příloze jsou úváděny jen původní, vytvořené zdrojové kódy.

1.1 Úloha komunikace dvou bodů

1.1.1 Primární vysílač

Main.c

```
#include <Nordic\reg24lu1.h>
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"
#include "cklf.h"

extern bool mode;

void main(void)
{
  initialize_sys();
  initialize_rad();

  while(true)
  {
   vysilej();
  }
}
```

Logic.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"
#include "cklf.h"

bool ls_byte = 0;
bool hs_byte = 0;
bool bin_pole[16];
uint16_t pocet = 0;
uint16_t pocet2 = 0;
uint16_t i;
extern bool set;
extern bool mode;
```

```
extern uint8_t stav;
extern uint8_t pload[1];
extern uint16_t count;
extern uint16_t timeout;

void vysilej(void)
{
  while(true)
  {

   if(TLACITKO1 != 1)
   {
     TIMER1_START();
     while(TLACITKO1 != 1)
       ;
     TIMER1_STOP();
     TL1 = 0xca;
     TH1 = 0xfa;
     count = 0;

     if(set == 1)
     {
       nastav(1);
     }
     else
     {
           prenos(CONTINUAL);

           for(i = 0;i < 1; i++)
           {

               fill_tx_fifo(255);
               stav = RADIO_PRACUJE;
               while(stav == RADIO_PRACUJE)
                 ;
               switch(stav)
             {
               case HAL_NRF_TX_DS:
                   LEDKA3_BLIK();
                   break;
                 case HAL_NRF_MAX_RT:
                   LEDKA2_BLIK();
                   break;
                 default:
                   break;
             }
           }
           RFCE = 0;
     }
   }
```

```
if(TLACITKO2 != 1)
{
  TIMER1_START();
  while(TLACITKO2 != 1)
    ;
  TIMER1_STOP();
  TL1 = 0xca;
  TH1 = 0xfa;
  count = 0;

  if(set == 1)
  {
      nastav(2);
  }
  else
  {
        prenos(CONTINUAL);

        for(i = 0;i < 256; i++)
        {

            fill_tx_fifo(127);
            stav = RADIO_PRACUJE;
            while(stav == RADIO_PRACUJE)
              ;
            switch(stav)
          {
            case HAL_NRF_TX_DS:
                LEDKA3_BLIK();
                break;
              case HAL_NRF_MAX_RT:
                LEDKA2_BLIK();
                break;
              default:
                break;
          }
            stav = RADIO_VOLNE;
        }
        RFCE = 0;
  }
}

if(TLACITKO3 != 1)
{
  TIMER1_START();
  while(TLACITKO3 != 1)
    ;
  TIMER1_STOP();
  TL1 = 0xca;
  TH1 = 0xfa;
  count = 0;
```

```
      if(set == 1)
      {
        nastav(3);
      }
      else
      {
            prenos(CONTINUAL);

            for(i = 0;i < 16384; i++)
            {
                fill_tx_fifo(63);
                stav = RADIO_PRACUJE;
                while(stav == RADIO_PRACUJE)
                  ;
                switch(stav)
              {
                case HAL_NRF_TX_DS:
                    LEDKA3_BLIK();
                    break;
                  case HAL_NRF_MAX_RT:
                    LEDKA2_BLIK();
                    break;
                  default:
                    break;
              }
                stav = RADIO_VOLNE;
                pocet2 = hal_nrf_get_transmit_attempts();
                pocet = pocet + pocet2 + 1;
            }
            RFCE = 0;
            ls_byte = 1;
            led_count(pocet);
            bin_led();
            blikej();
            set = 0;
            pocet = 0;
      }
     }
    }
}

void nastav(uint8_t conf)
{
  bool done = 0;

  while(done == 0)
  {
    switch(conf)
   {
     case 1:
```

```
        nastav_radio(posli_config(conf));
        LEDKA3_BLIK();
        done = 1;
        break;

      case 2:
        nastav_radio(posli_config(conf));
        LEDKA2_BLIK();
        done = 1;
        break;

      case 3:
        nastav_radio(posli_config(conf));
        LEDKA3_BLIK();
        done = 1;
        break;

      default:
        break;
    }
   }

  blikej();
  blikej();
  blikej();
  set = 0;
}

uint8_t posli_config(uint8_t conf)
{
  fill_tx_fifo(conf);
  prenos(ONE_BY_ONE);
  while(stav == RADIO_PRACUJE)
    ;
  switch(stav)
  {
    case HAL_NRF_TX_DS:
      break;
   case HAL_NRF_MAX_RT:
     break;
   default:
     break;
  }
  stav = RADIO_VOLNE;
  return conf;
}

void led_count(uint16_t pocet_prenosu)
{
  uint16_t p;
  bool z;
```

```
  uint8_t j;

  p = pocet_prenosu;

  for (j = 0; j < 16; j++)
  {
    z = p % 2;
   p = p / 2;

   bin_pole[j] = z;
  }
}

void bin_led(void)
{
  LEDKA1_OFF();
  LEDKA2_OFF();
  LEDKA3_OFF();

  while(ls_byte == 1 || hs_byte == 1)
  {
     while(ls_byte == 1)
     {
         if(TLACITKO1 != 1)
         {
            prepni_byty();
            if(bin_pole[5] == 1) LEDKA3_ON();
            else LEDKA3_OFF();
            if(bin_pole[6] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[7] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
         }
         else if(TLACITKO2 != 1)
         {
            prepni_byty();
            if(bin_pole[2] == 1) LEDKA3_ON();
            else LEDKA3_OFF();
            if(bin_pole[3] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[4] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
         }
         else if(TLACITKO3 != 1)
         {
            prepni_byty();
            LEDKA3_BLIK();
            if(bin_pole[0] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[1] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
```

```
        }
    }

    LEDKA1_OFF();
    LEDKA2_OFF();
    LEDKA3_OFF();

    while(hs_byte == 1)
    {
        if(TLACITKO1 != 1)
        {
           prepni_byty();
           if(bin_pole[14] == 1) LEDKA3_ON();
           else LEDKA3_OFF();
           if(bin_pole[15] == 1) LEDKA2_ON();
           else LEDKA2_OFF();

           LEDKA1_BLIK();
        }
        else if(TLACITKO2 != 1)
        {
           prepni_byty();
           if(bin_pole[11] == 1) LEDKA3_ON();
           else LEDKA3_OFF();
           if(bin_pole[12] == 1) LEDKA2_ON();
           else LEDKA2_OFF();
           if(bin_pole[13] == 1) LEDKA1_ON();
           else LEDKA1_OFF();
        }
        else if(TLACITKO3 != 1)
        {
           prepni_byty();
           if(bin_pole[8] == 1) LEDKA3_ON();
           else LEDKA3_OFF();
           if(bin_pole[9] == 1) LEDKA2_ON();
           else LEDKA2_OFF();
           if(bin_pole[10] == 1) LEDKA1_ON();
           else LEDKA1_OFF();
        }
    }

    LEDKA1_OFF();
    LEDKA2_OFF();
    LEDKA3_OFF();
  }
}

void prepni_byty(void)
{
  TIMER1_START();
  while(TLACITKO1 != 1 || TLACITKO2 != 1 || TLACITKO3 != 1)
```

```
  {
    if(TLACITKO1 != 1 && TLACITKO3 != 1)
   {
     ls_byte = 0;
     hs_byte = 0;
     TIMER1_STOP();
      TL1 = 0xca;
      TH1 = 0xfa;
      count = 0;
   }
  }
  TIMER1_STOP();
  TL1 = 0xca;
  TH1 = 0xfa;
  count = 0;
}
```

Config.c

```
#include "hal_nrf.h"
#include "config.h"
#include "cklf.h"

code       const       uint8_t       adresa[HAL_NRF_AW_5BYTES]       =
{0x22,0x33,0x44,0x55,0x02};

bool mode;
bool set = 0;
uint8_t stav;
uint8_t blik = 0;
uint8_t pload[1];
uint16_t count = 0;
uint16_t timeout = 15000;

extern bool ls_byte;
extern bool hs_byte;

void initialize_sys(void)
{
  P0ALT = 0x00;
  P0DIR = 0x38;
  P0    = 0x00;

  wait(15000);
  timer0();
  timer1();
  WUIRQ = 1;
  EA = 1;
  mode = HAL_NRF_PTX;
  WUCONF = 0x2F;
```

```
  blikej();
}

void initialize_rad(void)
{
  RFCKEN = 1;
  RFCTL = 0x10;

  hal_nrf_close_pipe(HAL_NRF_ALL);
  hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

  hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
  hal_nrf_set_auto_retr(3, 250);

  hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
  hal_nrf_set_address(HAL_NRF_TX, adresa);
  hal_nrf_set_address(HAL_NRF_PIPE0, adresa);

  switch_mode(mode, 44);

  hal_nrf_set_power_mode(HAL_NRF_PWR_UP);

  RF = 1;
  stav = RADIO_VOLNE;
}

void nastav_radio(uint8_t conf)
{
  switch(conf)
  {
    case 1:
     if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 0);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_OFF);
         hal_nrf_set_auto_retr(0, 250);
     }
     else
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
         hal_nrf_set_auto_retr(3, 250);
     }
     break;
   case 2:
     if(hal_nrf_get_datarate() == 0x00)
     {
       hal_nrf_set_datarate(HAL_NRF_2MBPS);
     }
```

```
     else
     {
       hal_nrf_set_datarate(HAL_NRF_1MBPS);
     }
     break;
   case 3:
     break;
   default:
     break;
  }
}

void switch_mode(bool mode, uint8_t channel)
{
  if(mode == HAL_NRF_PTX)
  {
    RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PTX);
   hal_nrf_set_rf_channel(channel);
  }
  if(mode == HAL_NRF_PRX)
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PRX);
   hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
   hal_nrf_set_rf_channel(channel);
   RFCE = 1;
  }
}

void fill_tx_fifo(uint8_t zprava)
{
  pload[0] = zprava;
  hal_nrf_write_tx_pload(pload, 1);
}

void prenos(bool zpusob)
{
  if(zpusob == 0)
  {
    CE_PULSE();
    stav = RADIO_PRACUJE;
  }
  else
  {
   CE_HIGH();
  }
}

void timer0(void)
{
```

```
  TL0 = 0xe9;
  TH0 = 0xcb;
  TMOD |= 0x01;
  TF0 = 0;
  ET0 = 1;
  TCON = 0;
  TR0 = 0;
}

void timer1(void)
{
  TL1 = 0xca;
  TH1 = 0xfa;
  TMOD |= 0x01;
  TF0 = 0;
  ET1 = 1;
  TCON = 0;
  TR1 = 0;
}

void timer0_interupt (void) interrupt 1
{
  TR0 = 0;
  TL0 = 0xE9;
  TH0 = 0xCB;

  switch(blik)
  {
    case LEDKA1:
      LEDKA1_OFF();
      break;
    case LEDKA2:
      LEDKA2_OFF();
      break;
    case LEDKA3:
      LEDKA3_OFF();
      break;
  }
  blik = 0;
}
void timer1_interupt (void) interrupt 3
{
  TL1 = 0xca;
  TH1 = 0xfa;
  count++;
  if(count == timeout)
  {
   set = 1;
   ls_byte = !ls_byte;
   hs_byte = !ls_byte;
   RFCE = 0;
```

```
   TR1 = 0;
   TL1 = 0xca;
    TH1 = 0xfa;
    count = 0;
   LEDKA1_ON();
   LEDKA2_ON();
   LEDKA3_ON();
  }
}

void radio_irq(void) interrupt 9
{
  switch(hal_nrf_get_clear_irq_flags())
  {
    case (1<<HAL_NRF_MAX_RT):
      hal_nrf_flush_tx();
      stav = HAL_NRF_MAX_RT;
      break;

    case (1<<HAL_NRF_TX_DS):
      stav = HAL_NRF_TX_DS;
      break;

    case (1<<HAL_NRF_RX_DR):
      hal_nrf_read_rx_pload(pload);
      stav = HAL_NRF_RX_DR;
      break;

    default:
      break;
  }
}

void wakeup_irg(void) interrupt 13
{

}

void blikej(void)
{
  uint8_t i;

  for(i = 0; i < 3; i++)
  {
    LEDKA1_BLIK();
    while(blik);
    wait(15000);
    LEDKA2_BLIK();
    while(blik);
    wait(15000);
   LEDKA3_BLIK();
```

```
    while(blik);
    wait(15000);
  }
}

void wait(uint16_t count_wait)
{
  uint16_t count_up;
  for(count_up = 0; count_up < count_wait; count_up ++);
}

void power_down(void)
{
  blikej();

  PWRDWN = 0x01;

  blikej();
}
```

1.1.2 Primární přijímač

Main.c

```
#include <Nordic\reg24lu1.h>
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

extern bool mode;
extern uint16_t pocet;

void main(void)
{
  initialize_sys();
  initialize_rad();

  while(true)
  {
   prijimej();
   blikej();
   led_count(pocet);
   bin_led();
   blikej();
   mode = HAL_NRF_PRX;
   RFCE = 1;
  }
}
```

Logic.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

bool ls_byte;
bool hs_byte;
bool bin_pole[16];

uint16_t pocet = 0;
uint8_t pocet2 = 0;

extern uint8_t stav;
extern bool mode;
extern uint8_t pload[1];
extern uint16_t count;

void prijimej(void)
{
  pocet = 0;
  pocet2 = 0;

  ziskej_status();

  while(mode == HAL_NRF_PRX)
  {
    while(TLACITKO1 != 1)
    {
      TIMER1_START();
    }
    while(TLACITKO2 != 1)
    {
      TIMER1_START();
    }
    TIMER1_STOP();
    TL1 = 0xe9;
     TH1 = 0xcb;
    count = 0;

    if(stav == HAL_NRF_RX_DR)
    {
         pocet2 = pload[0];

         if(pload[0] == 1)
         {
           nastav_radio(1);
           pocet = 0;
           pocet2 = 0;
         }
         else if(pload[0] == 2)
         {
           nastav_radio(2);
```

```
            pocet = 0;
            pocet2 = 0;
          }
          else if(pload[0] == 3)
          {
            nastav_radio(3);
            pocet = 0;
            pocet2 = 0;
          }
          else
          {
            LEDKA3_BLIK();
            pocet++;
          }
        stav = RADIO_VOLNE;
      }
    }
}

void led_count(uint16_t pocet_prenosu)
{
  uint16_t p;
  bool z;
  uint8_t j;

  p = pocet_prenosu;

  for (j = 0; j < 16; j++)
  {
    z = p % 2;
   p = p / 2;

   bin_pole[j] = z;
  }
}

void bin_led(void)
{
  LEDKA1_OFF();
  LEDKA2_OFF();
  LEDKA3_OFF();

  while(ls_byte == 1 || hs_byte == 1)
  {
     while(ls_byte == 1)
     {
         if(TLACITKO1 != 1)
         {
            prepni_byty();
            if(bin_pole[5] == 1) LEDKA3_ON();
            else LEDKA3_OFF();
```

```
        if(bin_pole[6] == 1) LEDKA2_ON();
        else LEDKA2_OFF();
        if(bin_pole[7] == 1) LEDKA1_ON();
        else LEDKA1_OFF();
    }
    else if(TLACITKO2 != 1)
    {
        prepni_byty();
        if(bin_pole[2] == 1) LEDKA3_ON();
        else LEDKA3_OFF();
        if(bin_pole[3] == 1) LEDKA2_ON();
        else LEDKA2_OFF();
        if(bin_pole[4] == 1) LEDKA1_ON();
        else LEDKA1_OFF();
    }
    else if(TLACITKO3 != 1)
    {
        prepni_byty();
        LEDKA3_BLIK();
        if(bin_pole[0] == 1) LEDKA2_ON();
        else LEDKA2_OFF();
        if(bin_pole[1] == 1) LEDKA1_ON();
        else LEDKA1_OFF();
    }
}

LEDKA1_OFF();
LEDKA2_OFF();
LEDKA3_OFF();

while(hs_byte == 1)
{
    if(TLACITKO1 != 1)
    {
        prepni_byty();
        if(bin_pole[14] == 1) LEDKA3_ON();
        else LEDKA3_OFF();
        if(bin_pole[15] == 1) LEDKA2_ON();
        else LEDKA2_OFF();

        LEDKA1_BLIK();
    }
    else if(TLACITKO2 != 1)
    {
        prepni_byty();
        if(bin_pole[11] == 1) LEDKA3_ON();
        else LEDKA3_OFF();
        if(bin_pole[12] == 1) LEDKA2_ON();
        else LEDKA2_OFF();
        if(bin_pole[13] == 1) LEDKA1_ON();
        else LEDKA1_OFF();
```

```
            }
            else if(TLACITKO3 != 1)
            {
               prepni_byty();
               if(bin_pole[8] == 1) LEDKA3_ON();
               else LEDKA3_OFF();
               if(bin_pole[9] == 1) LEDKA2_ON();
               else LEDKA2_OFF();
               if(bin_pole[10] == 1) LEDKA1_ON();
               else LEDKA1_OFF();
            }
         }

         LEDKA1_OFF();
         LEDKA2_OFF();
         LEDKA3_OFF();
      }
   }

   void prepni_byty(void)
   {
     TIMER1_START();
     while(TLACITKO1 != 1 || TLACITKO2 != 1 || TLACITKO3 != 1)
     {
       if(TLACITKO1 != 1 && TLACITKO3 != 1)
      {
        ls_byte = 0;
        hs_byte = 0;
        TIMER1_STOP();
         TL1 = 0xca;
         TH1 = 0xfa;
         count = 0;
      }
     }
     TIMER1_STOP();
     TL1 = 0xca;
     TH1 = 0xfa;
     count = 0;
   }
```

Config.c

```
   #include "hal_nrf.h"
   #include "config.h"
   #include "cklf.h"

   code       const       uint8_t       adresa[HAL_NRF_AW_5BYTES]       =
   {0x22,0x33,0x44,0x55,0x02};

   bool mode;
   uint8_t stav;
```

```
uint8_t blik = 0;
uint8_t pload[1];
uint16_t count = 0;
uint16_t timeout = 1000;

extern bool ls_byte;
extern bool hs_byte;

extern uint16_t pocet;
extern uint8_t pocet2;

void initialize_sys(void)
{
  P0ALT = 0x00;
  P0DIR = 0x38;
  P0    = 0x00;
  wait(15000);
  timer0();
  timer1();
  WUIRQ = 1;
  EA = 1;
  mode = HAL_NRF_PRX;
  WUCONF = 0xEF;
  blikej();
}

void initialize_rad(void)
{
  RFCKEN = 1;
  RFCTL = 0x10;

  hal_nrf_close_pipe(HAL_NRF_ALL);
  hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

  hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
  hal_nrf_set_auto_retr(3, 250);

  hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
  hal_nrf_set_address(HAL_NRF_TX, adresa);
  hal_nrf_set_address(HAL_NRF_PIPE0, adresa);
  hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);

  switch_mode(mode, 44);

  hal_nrf_set_power_mode(HAL_NRF_PWR_UP);

  RF = 1;
  stav = RADIO_VOLNE;
}

void nastav_radio(uint8_t conf)
```

```
{
  switch(conf)
  {
    case 1:
     if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 0);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_OFF);
         hal_nrf_set_auto_retr(0, 250);
         LEDKA1_ON();
     }
     else
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
         hal_nrf_set_auto_retr(3, 250);
         LEDKA1_OFF();
     }
     break;
   case 2:
     if(hal_nrf_get_datarate() == 0x00)
     {
       hal_nrf_set_datarate(HAL_NRF_2MBPS);
         LEDKA2_OFF();
     }
     else
     {
       hal_nrf_set_datarate(HAL_NRF_1MBPS);
         LEDKA2_ON();
     }
     break;
   case 3:
     break;
   default:
     break;
  }
}

void ziskej_status(void)
{
  if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
  {
   LEDKA1_OFF();
  }
  else
  {
   LEDKA2_ON();
  }
  if(hal_nrf_get_datarate() == 0x00)
```

```
  {
   LEDKA2_ON();
  }
  else
  {
   LEDKA2_OFF();
  }
}

void switch_mode(bool mode, uint8_t channel)
{
  if(mode == HAL_NRF_PTX)
  {
    RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PTX);
   hal_nrf_set_rf_channel(channel);
  }
  if(mode == HAL_NRF_PRX)
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PRX);
   hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
   hal_nrf_set_rf_channel(channel);
   RFCE = 1;
  }
}

void fill_tx_fifo(uint8_t zprava)
{
  pload[0] = zprava;
  hal_nrf_write_tx_pload(pload, 1);
}

void prenos(bool zpusob)
{
  if(zpusob == 0)
  {
    CE_PULSE();
    stav = RADIO_PRACUJE;
  }
  else
  {
   CE_HIGH();
  }
}

void timer0(void)
{
  TL0 = 0xe9;
  TH0 = 0xcb;
  TMOD |= 0x01;
```

```
  TF0 = 0;
  ET0 = 1;
  TCON = 0;
  TR0 = 0;
}

void timer1(void)
{
  TL1 = 0xe9;
  TH1 = 0xcb;
  TMOD |= 0x01;
  TF0 = 0;
  ET1 = 1;
  TCON = 0;
  TR1 = 0;
}

void timer0_interrupt (void) interrupt 1
{
  TR0 = 0;
  TL0 = 0xE9;
  TH0 = 0xCB;

  switch(blik)
  {
    case LEDKA1:
      LEDKA1_OFF();
      break;
    case LEDKA2:
      LEDKA2_OFF();
      break;
    case LEDKA3:
      LEDKA3_OFF();
      break;
  }
  blik = 0;
}
void timer1_interrupt (void) interrupt 3
{
  TL1 = 0xe9;
  TH1 = 0xcb;
  count++;
  if(count == timeout)
  {
    ls_byte = !ls_byte;
   hs_byte = !ls_byte;
   mode = HAL_NRF_PTX;
   RFCE = 0;
   TR1 = 0;
   TL1 = 0xe9;
    TH1 = 0xcb;
```

```
    count = 0;
   if(TLACITKO2 != 1)
   {
     pocet = pocet2;
   }
   LEDKA1_ON();
   LEDKA2_ON();
   LEDKA3_ON();
  }
}

void radio_interrupt(void) interrupt 9
{
  switch(hal_nrf_get_clear_irq_flags())
  {
    case (1<<HAL_NRF_MAX_RT):
      hal_nrf_flush_tx();
      stav = HAL_NRF_MAX_RT;
      break;

    case (1<<HAL_NRF_TX_DS):
      stav = HAL_NRF_TX_DS;
     stav = HAL_NRF_TX_DS;
      break;

    case (1<<HAL_NRF_RX_DR):
      hal_nrf_read_rx_pload(pload);
      stav = HAL_NRF_RX_DR;
      break;

    default:
      break;
  }
}

void blikej(void)
{
  uint8_t i;

  for(i = 0; i < 3; i++)
  {
    LEDKA1_BLIK();
    while(blik);
    wait(15000);
    LEDKA2_BLIK();
    while(blik);
    wait(15000);
   LEDKA3_BLIK();
    while(blik);
    wait(15000);
  }
```

```
}

void wait(uint16_t count_wait)
{
  uint16_t count_up;
  for(count_up = 0; count_up < count_wait; count_up ++);
}

void power_down(void)
{
  blikej();

  PWRDWN = 0x01;

  blikej();
}
```

1.2 Úloha komunikace tří bodů

1.2.1 Primární vysílač

Main.c

```
#include <Nordic\reg24lu1.h>
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

extern bool mode;

void main(void)
{
  initialize_sys();
  initialize_rad();

  while(true)
  {
   vysilej();
  }
}
```

Logic.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

bool ls_byte = 0;
bool hs_byte = 0;
bool bin_pole[16];
```

```
bool set = 0;
uint16_t pocet = 0;
uint16_t pocet2 = 0;
uint16_t pocet3 = 0;
uint16_t pocet4 = 0;

extern bool mode;
extern uint8_t stav;
extern uint8_t pload[1];
extern uint16_t count;
extern uint16_t timeout;

void vysilej(void)
{
  uint8_t conf;

  while(true)
  {
   pocet = 0;
   pocet2 = 0;
   LEDKA1_OFF();
   LEDKA2_OFF();
   LEDKA3_OFF();

   if(TLACITKO1 != 1 || TLACITKO2 != 1 || TLACITKO3 != 1)
   {
     if(TLACITKO1 != 1)
     {
       conf = 1;
         pocet = 1;
     }
     if(TLACITKO2 != 1)
     {
       conf = 2;
         pocet = 4;
     }
     if(TLACITKO3 != 1)
     {
       conf = 3;
         pocet = 512;
     }
     TIMER1_START();
     while(TLACITKO1 != 1 || TLACITKO2 != 1 || TLACITKO3 != 1)
       ;
     TIMER1_STOP();
     TL1 = 0xca;
     TH1 = 0xfa;
     count = 0;

     if(set == 1)
     {
```

```
      nastav(conf);
    }
    else
    {

        while(pocet2 != pocet)
        {
          pocet2++;

          posli_synch(READY);

          posli_data();

          posli_synch(NEXT);
        }
        ls_byte = 1;
        led_count(pocet3);
        bin_led();
        blikej();
        set = 0;
        pocet3 = 0;
    }
   LEDKA1_OFF();
   LEDKA2_OFF();
   LEDKA3_OFF();
   }
  }
}

void nastav(uint8_t conf)
{
  bool done = 0;

  while(done == 0)
  {
    switch(conf)
   {
     case 1:
       nastav_radio(posli_config(conf));
       LEDKA3_BLIK();
       done = 1;
       break;

     case 2:
       nastav_radio(posli_config(conf));
       LEDKA2_BLIK();
       done = 1;
       break;

     case 3:
       nastav_radio(posli_config(conf));
```

```
        LEDKA3_BLIK();
        done = 1;
        break;

      default:
        break;
    }
   }

   blikej();
   blikej();
   blikej();
   set = 0;
}

uint8_t posli_config(uint8_t conf)
{
   fill_tx_fifo(conf);
   prenos(ONE_BY_ONE);
   while(stav == RADIO_PRACUJE)
     ;
   stav = RADIO_VOLNE;
   switch_adress(1);
   fill_tx_fifo(conf);
   prenos(ONE_BY_ONE);
   while(stav == RADIO_PRACUJE)
     ;
   stav = RADIO_VOLNE;
   switch_adress(0);
   return conf;
}

void posli_synch(uint8_t synch)
{
   while(stav != HAL_NRF_TX_DS)
   {
     fill_tx_fifo(synch);
     prenos(ONE_BY_ONE);
    while(stav == RADIO_PRACUJE)
      ;
    switch(stav)
     {
       case HAL_NRF_TX_DS:
        break;
      case HAL_NRF_MAX_RT:
        LEDKA3_BLIK();
        break;
      default:
        break;
     }
    if(TLACITKO1 != 1)
```

```
   {
     if(TLACITKO3 != 1)
     {
          pocet2 = pocet;
          stav = HAL_NRF_TX_DS;
          while(TLACITKO1 != 1 || TLACITKO3 != 1)
            ;
     }
   }
  }
  stav = RADIO_VOLNE;
}

void posli_data(void)
{
  uint16_t i;
  prenos(CONTINUAL);

  for(i = 0;i < 32; i++)
  {
   fill_tx_fifo(i + 10);
   stav = RADIO_PRACUJE;
   while(stav == RADIO_PRACUJE)
     ;
   switch(stav)
    {
      case HAL_NRF_TX_DS:
       LEDKA3_ON();
       break;
     case HAL_NRF_MAX_RT:
       LEDKA2_ON();
       break;
     default:
       break;
    }
   stav = RADIO_VOLNE;
   LEDKA2_OFF();
   LEDKA3_OFF();
   pocet4 = hal_nrf_get_transmit_attempts();
   pocet3 = pocet3 + pocet4 + 1;
  }
  RFCE = 0;
}

void led_count(uint16_t pocet_prenosu)
{
  uint16_t p;
  bool z;
  uint8_t j;

  p = pocet_prenosu;
```

```
  for (j = 0; j < 16; j++)
  {
    z = p % 2;
   p = p / 2;

   bin_pole[j] = z;
  }
}

void bin_led(void)
{
  LEDKA1_OFF();
  LEDKA2_OFF();
  LEDKA3_OFF();

  while(ls_byte == 1 || hs_byte == 1)
  {
     while(ls_byte == 1)
     {
         if(TLACITKO1 != 1)
         {
            prepni_byty();
            if(bin_pole[5] == 1) LEDKA3_ON();
            else LEDKA3_OFF();
            if(bin_pole[6] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[7] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
         }
         else if(TLACITKO2 != 1)
         {
            prepni_byty();
            if(bin_pole[2] == 1) LEDKA3_ON();
            else LEDKA3_OFF();
            if(bin_pole[3] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[4] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
         }
         else if(TLACITKO3 != 1)
         {
            prepni_byty();
            LEDKA3_BLIK();
            if(bin_pole[0] == 1) LEDKA2_ON();
            else LEDKA2_OFF();
            if(bin_pole[1] == 1) LEDKA1_ON();
            else LEDKA1_OFF();
         }
     }
```

```
      LEDKA1_OFF();
      LEDKA2_OFF();
      LEDKA3_OFF();

      while(hs_byte == 1)
      {
          if(TLACITKO1 != 1)
          {
             prepni_byty();
             if(bin_pole[14] == 1) LEDKA3_ON();
             else LEDKA3_OFF();
             if(bin_pole[15] == 1) LEDKA2_ON();
             else LEDKA2_OFF();

             LEDKA1_BLIK();
          }
          else if(TLACITKO2 != 1)
          {
             prepni_byty();
             if(bin_pole[11] == 1) LEDKA3_ON();
             else LEDKA3_OFF();
             if(bin_pole[12] == 1) LEDKA2_ON();
             else LEDKA2_OFF();
             if(bin_pole[13] == 1) LEDKA1_ON();
             else LEDKA1_OFF();
          }
          else if(TLACITKO3 != 1)
          {
             prepni_byty();
             if(bin_pole[8] == 1) LEDKA3_ON();
             else LEDKA3_OFF();
             if(bin_pole[9] == 1) LEDKA2_ON();
             else LEDKA2_OFF();
             if(bin_pole[10] == 1) LEDKA1_ON();
             else LEDKA1_OFF();
          }
      }

      LEDKA1_OFF();
      LEDKA2_OFF();
      LEDKA3_OFF();
  }
}

void prepni_byty(void)
{
  TIMER1_START();
  while(TLACITKO1 != 1 || TLACITKO2 != 1 || TLACITKO3 != 1)
  {
    if(TLACITKO1 != 1 && TLACITKO3 != 1)
   {
```

```
      ls_byte = 0;
      hs_byte = 0;
      TIMER1_STOP();
       TL1 = 0xca;
       TH1 = 0xfa;
       count = 0;
   }
  }
  TIMER1_STOP();
  TL1 = 0xca;
  TH1 = 0xfa;
  count = 0;
}
```

Config.c

```
#include "hal_nrf.h"
#include "config.h"
#include "cklf.h"

code       const       uint8_t       adresa[HAL_NRF_AW_5BYTES]       =
{0x22,0x33,0x44,0x55,0x01};
code       const       uint8_t       adresa2[HAL_NRF_AW_5BYTES]       =
{0x22,0x33,0x44,0x55,0x02};

bool mode;
uint8_t stav;
uint8_t blik = 0;
uint8_t pload[1];
uint16_t count = 0;
uint16_t timeout = 10000;

extern bool ls_byte;
extern bool hs_byte;
extern bool set;

void initialize_sys(void)
{
  P0ALT = 0x00;
  P0DIR = 0x38;
  P0    = 0x00;
  wait(15000);
  timer0();
  timer1();
  WUIRQ = 1;
  EA = 1;
  mode = HAL_NRF_PTX;
  WUCONF = 0xEF;
  blikej();
}
```

```
void initialize_rad(void)
{
  RFCKEN = 1;
  RFCTL = 0x10;

  hal_nrf_close_pipe(HAL_NRF_ALL);
  hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

  hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
  hal_nrf_set_auto_retr(3, 250);

  hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
  hal_nrf_set_address(HAL_NRF_TX, adresa);
  hal_nrf_set_address(HAL_NRF_PIPE0, adresa);

  switch_mode(mode, 44);

  hal_nrf_set_power_mode(HAL_NRF_PWR_UP);

  RF = 1;
  stav = RADIO_VOLNE;
}

void nastav_radio(uint8_t conf)
{
  switch(conf)
  {
    case 1:
     if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 0);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_OFF);
         hal_nrf_set_auto_retr(0, 250);
     }
     else
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
         hal_nrf_set_auto_retr(3, 250);
     }
     break;
   case 2:
     if(hal_nrf_get_datarate() == 0x00)
     {
       hal_nrf_set_datarate(HAL_NRF_2MBPS);
     }
     else
     {
       hal_nrf_set_datarate(HAL_NRF_1MBPS);
```

```
      }
      break;
    case 3:
      break;
    default:
      break;
  }
}

void switch_adress(bool adr)
{
  if(adr == 0)
  {
    hal_nrf_set_address(HAL_NRF_TX, adresa);
  }
  else
  {
    hal_nrf_set_address(HAL_NRF_TX, adresa2);
  }
}

void switch_mode(bool mode, uint8_t channel)
{
  if(mode == HAL_NRF_PTX)
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PTX);
   hal_nrf_set_rf_channel(channel);
  }
  if(mode == HAL_NRF_PRX)
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PRX);
   hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
   hal_nrf_set_rf_channel(channel);
   RFCE = 1;
  }
}

void fill_tx_fifo(uint8_t zprava)
{
  pload[0] = zprava;
  hal_nrf_write_tx_pload(pload, 1);
}

void prenos(bool zpusob)
{
  if(zpusob == 0)
  {
    CE_PULSE();
    stav = RADIO_PRACUJE;
```

```
  }
  else
  {
   CE_HIGH();
  }
}

void timer0(void)
{
  TL0 = 0xe9;
  TH0 = 0xcb;
  TMOD |= 0x01;
  TF0 = 0;
  ET0 = 1;
  TCON = 0;
  TR0 = 0;
}

void timer1(void)
{
  TL1 = 0xca;
  TH1 = 0xfa;
  TMOD |= 0x01;
  TF0 = 0;
  ET1 = 1;
  TCON = 0;
  TR1 = 0;
}

void timer0_interupt (void) interrupt 1
{
  TR0 = 0;
  TL0 = 0xE9;
  TH0 = 0xCB;

  switch(blik)
  {
    case LEDKA1:
      LEDKA1_OFF();
      break;
    case LEDKA2:
      LEDKA2_OFF();
      break;
    case LEDKA3:
      LEDKA3_OFF();
      break;
  }
  blik = 0;
}
void timer1_interupt (void) interrupt 3
{
```

```
  TL1 = 0xca;
  TH1 = 0xfa;
  count++;
  if(count == timeout)
  {
   set = 1;
   RFCE = 0;
   TR1 = 0;
   TL1 = 0xca;
   TH1 = 0xfa;
   count = 0;
   ls_byte = !ls_byte;
   hs_byte = !ls_byte;
   LEDKA1_ON();
   LEDKA2_ON();
   LEDKA3_ON();
  }
}

void radio_irq(void) interrupt 9
{
  switch(hal_nrf_get_clear_irq_flags())
  {
    case (1<<HAL_NRF_MAX_RT):
      hal_nrf_flush_tx();
      stav = HAL_NRF_MAX_RT;
      break;

    case (1<<HAL_NRF_TX_DS):
      stav = HAL_NRF_TX_DS;
      break;

    case (1<<HAL_NRF_RX_DR):
      hal_nrf_read_rx_pload(pload);
      stav = HAL_NRF_RX_DR;
      break;

    default:
      break;
  }
}

void blikej(void)
{
  uint8_t i;

  for(i = 0; i < 3; i++)
  {
    LEDKA1_BLIK();
    while(blik);
    wait(15000);
```

```
    LEDKA2_BLIK();
    while(blik);
    wait(15000);
   LEDKA3_BLIK();
    while(blik);
    wait(15000);
  }
}

void wait(uint16_t count_wait)
{
  uint16_t count_up;
  for(count_up = 0; count_up < count_wait; count_up ++);
}
```

1.2.2 Střední komunikační bod

Main.c

```
#include <Nordic\reg24lu1.h>
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

extern bool mode;
/*extern bool transmit;
extern uint16_t pocet2;
extern uint16_t pocet;*/

void main(void)
{
  initialize_sys();
  initialize_rad();

  while(true)
  {
   prijimej();
   switch_mode(mode, 44);
   vysilej();
   switch_mode(mode, 44);
  }
}
```

Logic.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "config.h"

bool stav_pamet = 0;
uint8_t index = 0;
uint8_t pamet[32];
```

```
uint16_t pocet = 0;

extern bool mode;
extern uint8_t stav;
extern uint8_t pload[1];
extern uint16_t count;
extern uint16_t timeout;

void prijimej(void)
{
  index = 0;

  while(mode == HAL_NRF_PRX)
  {
   if(stav == HAL_NRF_RX_DR)
   {
     switch(pload[0])
     {
       case READY:
           break;

         case NEXT:
           if(stav_pamet == PAMET_PLNA)
           {
             mode = HAL_NRF_PTX;
               index = 0;
           }
           pload[0] = 0;
           break;

         case 1:
           nastav_radio(1);
           break;

         case 2:
           nastav_radio(2);
           break;

         case 3:
           nastav_radio(3);
           break;

         default:
           pamet[index] = pload[0];
           //pload[0] = 0;
           //pocet++;
           index++;
           pocet++;
           stav_pamet = PAMET_PLNA;
           break;
     }
```

```
      stav = RADIO_VOLNE;
     }
    }
  }

  void vysilej()
  {
    uint8_t i;

    wait(150);

    prenos(CONTINUAL);

    for(i = 0;i < pocet; i++)
    {
     fill_tx_fifo(pamet[i]);
     stav = RADIO_PRACUJE;
     while(stav == RADIO_PRACUJE)
       ;
     switch(stav)
     {
       case HAL_NRF_TX_DS:
           LEDKA3_BLIK();
           break;
       case HAL_NRF_MAX_RT:
           LEDKA2_BLIK();
           break;
       default:
           break;
     }
     pamet[i] = 0;
     stav = RADIO_VOLNE;
    }
    stav_pamet = PAMET_PRAZDNA;
    pocet = 0;
    mode = HAL_NRF_PRX;
  }
```

Config.c

```
  #include "hal_nrf.h"
  #include "config.h"
  #include "cklf.h"

  code       const       uint8_t       adresa[HAL_NRF_AW_5BYTES]       =
  {0x22,0x33,0x44,0x55,0x01};
  code       const       uint8_t       adresa2[HAL_NRF_AW_5BYTES]       =
  {0x22,0x33,0x44,0x55,0x02};

  bool mode;
  uint8_t stav;
```

```
uint8_t blik = 0;
uint8_t pload[1];
uint16_t count = 0;
uint16_t timeout = 1000;

void initialize_sys(void)
{
  P0ALT = 0x00;
  P0DIR = 0x38;
  P0    = 0x00;
  wait(15000);
  timer0();
  //timer1();
  WUIRQ = 1;
  EA = 1;
  mode = HAL_NRF_PRX;
  WUCONF = 0xEF;
}

void initialize_rad(void)
{
  RFCKEN = 1;
  RFCTL = 0x10;

  hal_nrf_close_pipe(HAL_NRF_ALL);
  hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

  hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
  hal_nrf_set_auto_retr(3, 250);

  hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
  hal_nrf_set_address(HAL_NRF_TX, adresa);
  hal_nrf_set_address(HAL_NRF_PIPE0, adresa);

  switch_mode(mode, 44);

  hal_nrf_set_power_mode(HAL_NRF_PWR_UP);

  RF = 1;
  stav = RADIO_VOLNE;
}

void nastav_radio(uint8_t conf)
{
  switch(conf)
  {
    case 1:
     if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 0);
```

```
          hal_nrf_set_crc_mode(HAL_NRF_CRC_OFF);
          hal_nrf_set_auto_retr(0, 250);
          LEDKA1_ON();
      }
      else
      {
        hal_nrf_close_pipe(HAL_NRF_ALL);
          hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);
          hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
          hal_nrf_set_auto_retr(3, 250);
          LEDKA1_OFF();
      }
      break;
    case 2:
      if(hal_nrf_get_datarate() == 0x00)
      {
        hal_nrf_set_datarate(HAL_NRF_2MBPS);
          LEDKA2_OFF();
      }
      else
      {
        hal_nrf_set_datarate(HAL_NRF_1MBPS);
          LEDKA2_ON();
      }
      break;
    case 3:
      break;
    default:
      break;
  }
}

void switch_mode(bool mode, uint8_t channel)
{
  if(mode == HAL_NRF_PTX)
  {
    RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PTX);
   hal_nrf_set_rf_channel(channel);
   hal_nrf_set_address(HAL_NRF_TX, adresa2);
  }
  if(mode == HAL_NRF_PRX)
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PRX);
   hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
   hal_nrf_set_rf_channel(channel);
   hal_nrf_set_address(HAL_NRF_TX, adresa);
   RFCE = 1;
  }
}
```

```
void fill_tx_fifo(uint8_t zprava)
{
  pload[0] = zprava;
  hal_nrf_write_tx_pload(pload, 1);
}

void prenos(bool zpusob)
{
  if(zpusob == 0)
  {
    CE_PULSE();
    stav = RADIO_PRACUJE;
  }
  else
  {
   CE_HIGH();
  }
}

void timer0(void)
{
  TL0 = 0xe9;
  TH0 = 0xcb;
  TMOD |= 0x01;
  TF0 = 0;
  ET0 = 1;
  TCON = 0;
  TR0 = 0;
}

void timer1(void)
{
  TL1 = 0xca;
  TH1 = 0xfa;
  TMOD |= 0x01;
  TF0 = 0;
  ET1 = 1;
  TCON = 0;
  TR1 = 0;
}

void timer0_interupt (void) interrupt 1
{
  TR0 = 0;
  TL0 = 0xE9;
  TH0 = 0xCB;

  switch(blik)
  {
    case LEDKA1:
```

```
      LEDKA1_OFF();
      break;
    case LEDKA2:
      LEDKA2_OFF();
      break;
    case LEDKA3:
      LEDKA3_OFF();
      break;
  }
  blik = 0;
}
void timer1_interupt (void) interrupt 3
{

}

void radio_irq(void) interrupt 9
{
  switch(hal_nrf_get_clear_irq_flags())
  {
    case (1<<HAL_NRF_MAX_RT):
      hal_nrf_flush_tx();
      stav = HAL_NRF_MAX_RT;
      break;

    case (1<<HAL_NRF_TX_DS):
      stav = HAL_NRF_TX_DS;
      break;

    case (1<<HAL_NRF_RX_DR):
      hal_nrf_read_rx_pload(pload);
      stav = HAL_NRF_RX_DR;
      break;

    default:
      break;
  }
}

void blikej(void)
{
  uint8_t i;

  for(i = 0; i < 3; i++)
  {
    LEDKA1_BLIK();
    while(blik);
    wait(15000);
    LEDKA2_BLIK();
    while(blik);
    wait(15000);
```

```
   LEDKA3_BLIK();
    while(blik);
    wait(15000);
  }
}

void wait(uint16_t count_wait)
{
  uint16_t count_up;
  for(count_up = 0; count_up < count_wait; count_up ++);
}
```

1.2.3 Primární přijímač

Main.c

```
#include <Nordic\reg24lu1.h>
#include "hal_nrf.h"
#include "logic.h"
#include "configure.h"

void main(void)
{
  initialize_sys();
  initialize_rad();
  while(true)
  {
   prijimej();
  }
}
```

Logic.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "configure.h"
#include "usbfces.h"
#include "nordic_common.h"

bool ls_byte;
bool hs_byte;
bool bin_pole[16];
uint8_t index = 0;
uint8_t cislo[2];
uint8_t pamet[32];
uint16_t pocet = 0;
uint16_t pocet2 = 0;

extern uint8_t stav;
extern bool mode;
extern uint8_t pload[1];
```

```
extern uint16_t count;

void prijimej(void)
{
  pocet = 0;
  pocet2 = 0;
  index = 0;

  ziskej_status();


  while(mode == HAL_NRF_PRX)
  {
    LEDKA1_OFF();
    LEDKA3_OFF();
    if(TLACITKO1 != 1)
    {
      while(TLACITKO1 != 1)
        ;
      cislo[1] = LSB(pocet);
      cislo[0] = MSB(pocet);
      usb_posli_paket2(&cislo);
    }
    if(TLACITKO2 != 1)
    {
      while(TLACITKO2 != 1)
        ;
      pocet = 0;
    }

    if(stav == HAL_NRF_RX_DR)
    {
         switch(pload[0])
         {
           case 1:
             nastav_radio(1);
               break;

           case 2:
             nastav_radio(2);
               break;

           case 3:
             nastav_radio(3);
               break;

           default:
               LEDKA3_ON();
               pocet++;
               pamet[index] = pload[0];
               //usb_posli_paket1(&pload[0]);
```

```
                index++;

                pload[0] = 0;
                if(index == 32)
                {
                  //uint8_t j;

                  uint16_t vysledek = 0;
                  index = 0;
                  //cislo[1] = LSB(pocet);
                  //cislo[0] = MSB(pocet);
                  //usb_posli_paket2(&cislo);
                  if(pocet == 128)
                  {
                    LEDKA1_ON();
                  }
                }
                break;
          }
          stav = RADIO_VOLNE;
      }
    }
  }
```

Configure.c

```
  #include "hal_nrf.h"
  #include "configure.h"
  #include "usbfces.h"

  code       const       uint8_t       adresa[HAL_NRF_AW_5BYTES]       =
  {0x22,0x33,0x44,0x55,0x02};

  bool mode;
  uint8_t stav;
  uint8_t blik = 0;
  uint8_t pload[1];
  uint16_t count = 0;
  uint16_t timeout = 1000;

  extern bool ls_byte;
  extern bool hs_byte;

  void initialize_sys(void)
  {
    P0ALT = 0x00;
    P0DIR = 0x38;
    P0    = 0x00;
    wait(15000);
    timer0();
    timer1();
```

```
  WUIRQ = 1;
  initialize_usb();
  EA = 1;
  usb_config_wait();
  mode = HAL_NRF_PRX;
  WUCONF = 0xEF;
  blikej();
}

void initialize_rad(void)
{
  RFCKEN = 1;
  RFCTL = 0x10;

  hal_nrf_close_pipe(HAL_NRF_ALL);
  hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);

  hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
  hal_nrf_set_auto_retr(3, 250);

  hal_nrf_set_address_width(HAL_NRF_AW_5BYTES);
  hal_nrf_set_address(HAL_NRF_TX, adresa);
  hal_nrf_set_address(HAL_NRF_PIPE0, adresa);

  switch_mode(mode, 44);

  hal_nrf_set_power_mode(HAL_NRF_PWR_UP);

  RF = 1;
  stav = RADIO_VOLNE;
}

void nastav_radio(uint8_t conf)
{
  switch(conf)
  {
    case 1:
     if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 0);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_OFF);
         hal_nrf_set_auto_retr(0, 250);
         LEDKA1_ON();
     }
     else
     {
       hal_nrf_close_pipe(HAL_NRF_ALL);
         hal_nrf_open_pipe(HAL_NRF_PIPE0, 1);
         hal_nrf_set_crc_mode(HAL_NRF_CRC_16BIT);
         hal_nrf_set_auto_retr(3, 250);
```

```
        LEDKA1_OFF();
      }
      break;
    case 2:
      if(hal_nrf_get_datarate() == 0x00)
      {
        hal_nrf_set_datarate(HAL_NRF_2MBPS);
          LEDKA2_OFF();
      }
      else
      {
        hal_nrf_set_datarate(HAL_NRF_1MBPS);
          LEDKA2_ON();
      }
      break;
    case 3:
      break;
    default:
      break;
  }
}

void ziskej_status(void)
{
  if(hal_nrf_get_pipe_status(HAL_NRF_PIPE0) == 0x03)
  {
   LEDKA1_OFF();
  }
  else
  {
   LEDKA2_ON();
  }
  if(hal_nrf_get_datarate() == 0x00)
  {
   LEDKA2_ON();
  }
  else
  {
   LEDKA2_OFF();
  }
}

void switch_mode(bool mode, uint8_t channel)
{
  if(mode == HAL_NRF_PTX)
  {
    RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PTX);
   hal_nrf_set_rf_channel(channel);
  }
  if(mode == HAL_NRF_PRX)
```

```
  {
   RFCE = 0;
   hal_nrf_set_operation_mode(HAL_NRF_PRX);
   hal_nrf_set_rx_pload_width(HAL_NRF_PIPE0, 1);
   hal_nrf_set_rf_channel(channel);
   RFCE = 1;
  }
}

void fill_tx_fifo(uint8_t zprava)
{
  pload[0] = zprava;
  hal_nrf_write_tx_pload(pload, 1);
}

void prenos(bool zpusob)
{
  if(zpusob == 0)
  {
    CE_PULSE();
    stav = RADIO_PRACUJE;
  }
  else
  {
   CE_HIGH();
  }
}

void timer0(void)
{
  TL0 = 0xe9;
  TH0 = 0xcb;
  TMOD |= 0x01;
  TF0 = 0;
  ET0 = 1;
  TCON = 0;
  TR0 = 0;
}

void timer1(void)
{
  TL1 = 0xca;
  TH1 = 0xfa;
  TMOD |= 0x01;
  TF0 = 0;
  ET1 = 1;
  TCON = 0;
  TR1 = 0;
}

void timer0_interrupt (void) interrupt 1
```

```
{
  TR0 = 0;
  TL0 = 0xE9;
  TH0 = 0xCB;

  switch(blik)
  {
    case LEDKA1:
      LEDKA1_OFF();
      break;
    case LEDKA2:
      LEDKA2_OFF();
      break;
    case LEDKA3:
      LEDKA3_OFF();
      break;
  }
  blik = 0;
}
void timer1_interrupt (void) interrupt 3
{
  TL1 = 0xe9;
  TH1 = 0xcb;
  count++;
  if(count == timeout)
  {
    ls_byte = !ls_byte;
   hs_byte = !ls_byte;
   mode = HAL_NRF_PTX;
   RFCE = 0;
   TR1 = 0;
   TL1 = 0xe9;
    TH1 = 0xcb;
    count = 0;
   if(TLACITKO2 != 1)
   {
     //pocet = pocet2;
   }
   LEDKA1_OFF();
   LEDKA2_OFF();
   LEDKA3_OFF();
  }
}

void radio_interrupt(void) interrupt 9
{
  switch(hal_nrf_get_clear_irq_flags())
  {
    case (1<<HAL_NRF_MAX_RT):
      hal_nrf_flush_tx();
      stav = HAL_NRF_MAX_RT;
```

```
      break;

    case (1<<HAL_NRF_TX_DS):
      stav = HAL_NRF_TX_DS;
     stav = HAL_NRF_TX_DS;
      break;

    case (1<<HAL_NRF_RX_DR):
      hal_nrf_read_rx_pload(pload);
      stav = HAL_NRF_RX_DR;
      break;

    default:
      break;
  }
}

void blikej(void)
{
  uint8_t i;

  for(i = 0; i < 3; i++)
  {
    LEDKA1_BLIK();
    while(blik);
    wait(15000);
    LEDKA2_BLIK();
    while(blik);
    wait(15000);
   LEDKA3_BLIK();
    while(blik);
    wait(15000);
  }
}

void wait(uint16_t count_wait)
{
  uint16_t count_up;
  for(count_up = 0; count_up < count_wait; count_up ++);
}
```

Usbfces.c

```
#include "hal_nrf.h"
#include "logic.h"
#include "configure.h"
#include "hal_usb.h"
#include "hal_usb_hid.h"
#include "nordic_common.h"
#include "usbfces.h"
```

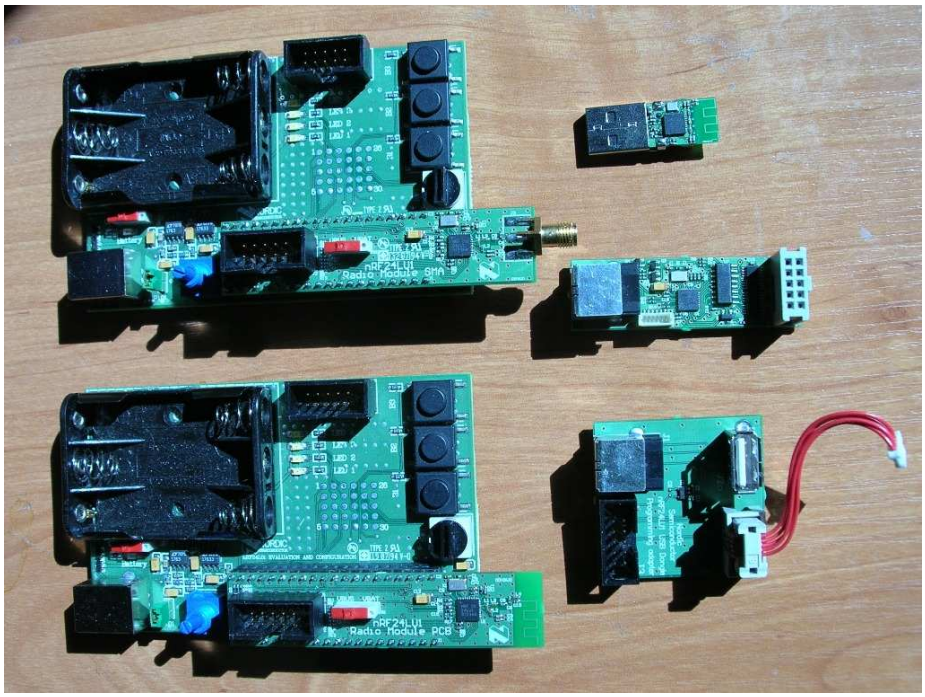
```
static   hal_usb_dev_req_resp_t   callback_device_req(hal_usb_device_req*
req, uint8_t** data_ptr, uint16_t* size) reentrant;
static void callback_suspend(uint8_t allow_remote_wu) reentrant;
static void callback_resume() reentrant;
static void callback_reset() reentrant;
static  uint8_t  callback_endpoint1_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant;
static  uint8_t  callback_endpoint2_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant;

volatile uint8_t stav_usb;
bool endpoint1;
bool endpoint2;

void initialize_usb(void)
{
  hal_usb_init(true,        callback_device_req,        callback_suspend,
callback_resume, callback_reset);

  hal_usb_endpoint_config(0x81, 1, callback_endpoint1_in);
  hal_usb_endpoint_config(0x82, 2, callback_endpoint2_in);

  endpoint1 = true;
  endpoint2 = true;
  stav_usb = AWAKE;
}

void wu_usb(void)
{
  hal_usb_wakeup();
  stav_usb = AWAKE;
}

uint8_t dej_stav_usb(void)
{
  return stav_usb;
}

void usb_config_wait(void)
{
  volatile hal_usb_state_t stav2_usb;
  do
  {
    stav2_usb = hal_usb_get_state();
  }
  while(stav2_usb != CONFIGURED);
  if(stav2_usb == CONFIGURED)
  {
    LEDKA1_BLIK();
  }
}
```

```
hal_usb_state_t ziskej_stav(void)
{
  volatile hal_usb_state_t stav3_usb;
  stav3_usb = hal_usb_get_state();
  return stav3_usb;
}

void usb_posli_paket1(uint8_t* in_data)
{
  while(!endpoint1)
  {
    //LEDKA2_BLIK();
  }
  endpoint1 = false;

  hal_usb_send_data(0x81, in_data, 1);

  LEDKA2_BLIK();
}

void usb_posli_paket2(uint8_t* in_data)
{
  while(!endpoint2)
  {
    //LEDKA2_BLIK();
  }
  endpoint2 = false;

  hal_usb_send_data(0x82, in_data, 2);

  LEDKA2_BLIK();
}

static   hal_usb_dev_req_resp_t   callback_device_req(hal_usb_device_req*
req, uint8_t** data_ptr, uint16_t* size) reentrant
{
    hal_usb_dev_req_resp_t retval;

    if(  hal_usb_hid_device_req_proc(req,  data_ptr,  size,  &retval)  ==
true )
    {
        LEDKA2_BLIK();
        return retval;
    }
    else
    {
        LEDKA3_BLIK();
         return STALL;
    }
    return STALL;
}
```

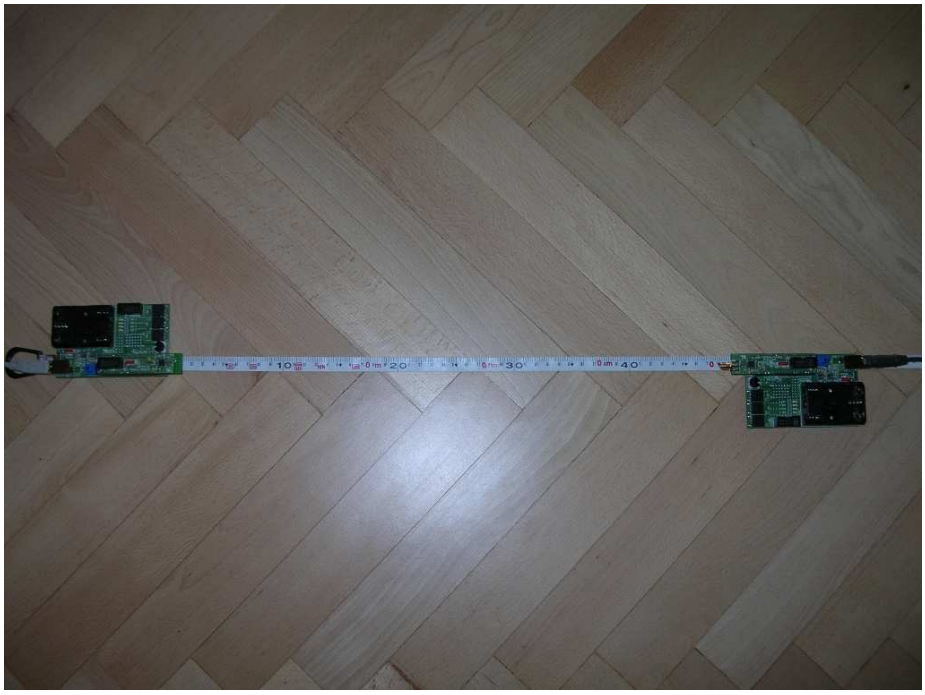
```
static void callback_suspend(uint8_t allow_remote_wu) reentrant
{
  USBSLP = 1;
  LEDKA2_BLIK();
  if (allow_remote_wu == 1)
  {
    WUCONF = (BIT_5 | BIT_3 | BIT_1);
    stav_usb = WU_ON;
  }
  else
  {
    WUCONF = (BIT_3 | BIT_1);
    stav_usb = WU_OFF;
  }
}

static void callback_resume() reentrant
{
  endpoint1 = true;
  endpoint2 = true;
  stav_usb = AWAKE;
  LEDKA2_BLIK();
}

static void callback_reset() reentrant
{
  endpoint1 = true;
  endpoint2 = true;
  stav_usb = AWAKE;
  LEDKA2_BLIK();
}

static  uint8_t  callback_endpoint1_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant
{
  endpoint1 = true;
  return 0x60;
  LEDKA2_BLIK();
}
static  uint8_t  callback_endpoint2_in(uint8_t*  adr_ptr,  uint8_t*  size)
reentrant
{
  endpoint2 = true;
  return 0x60;
  LEDKA2_BLIK();
}
```

2 Obrazové přílohy

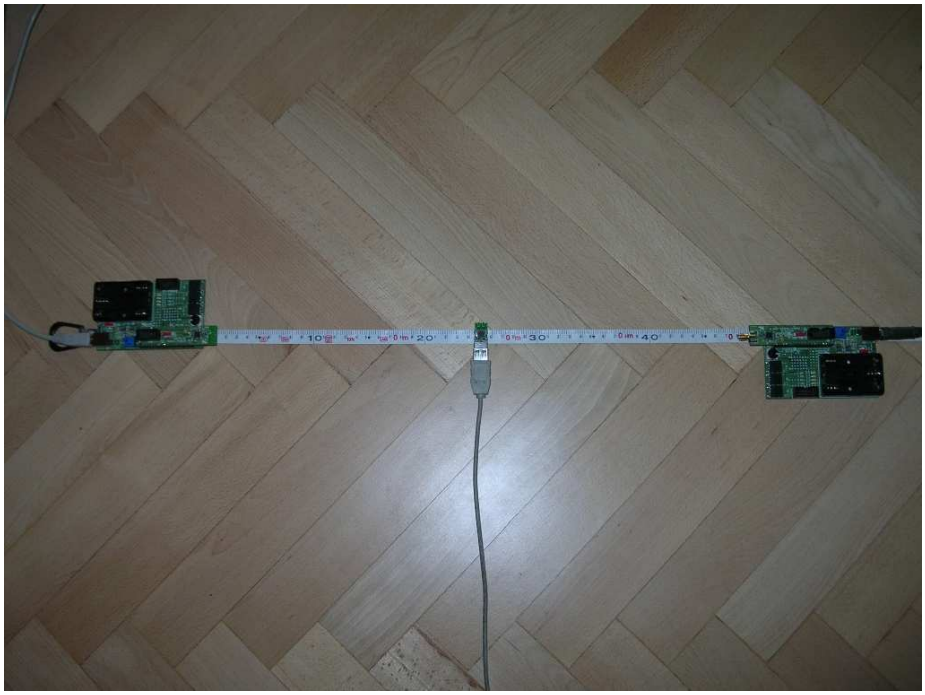
2.1 Vývojový kit nRF24LU1



Obrazová příloha č. 2 – Vývojový kit nRF24LU1.



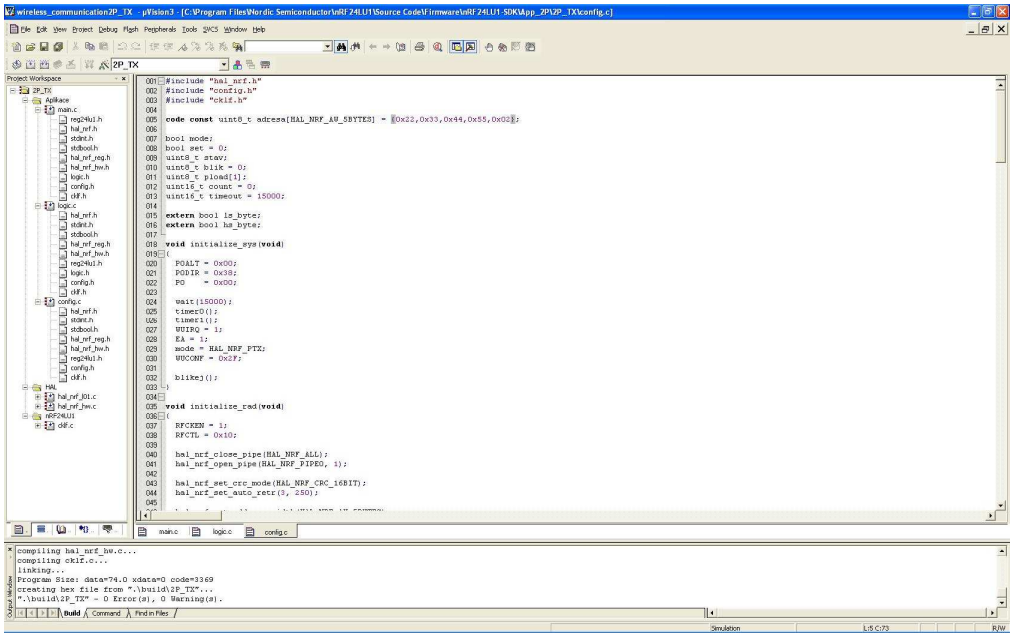
Obrazová příloha č. 2 – Test přenosu aplikace kom. 2 body.



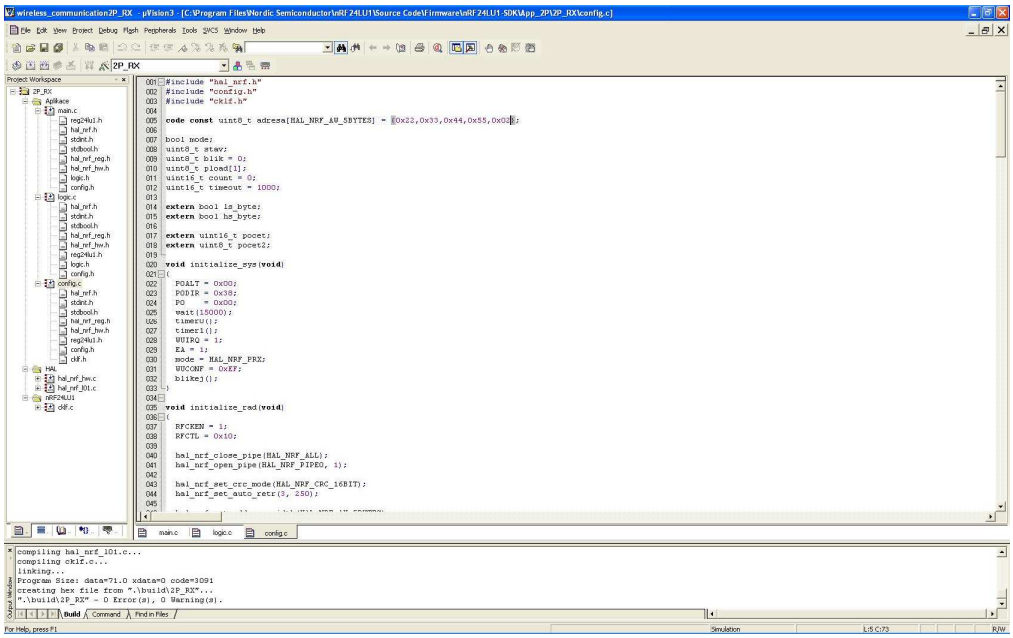
Obrazová příloha č. 3 – Test přenosu aplikace kom. 3 body.

2.2 Keil µVision

2.2.1 Úloha komunikace dvou bodů

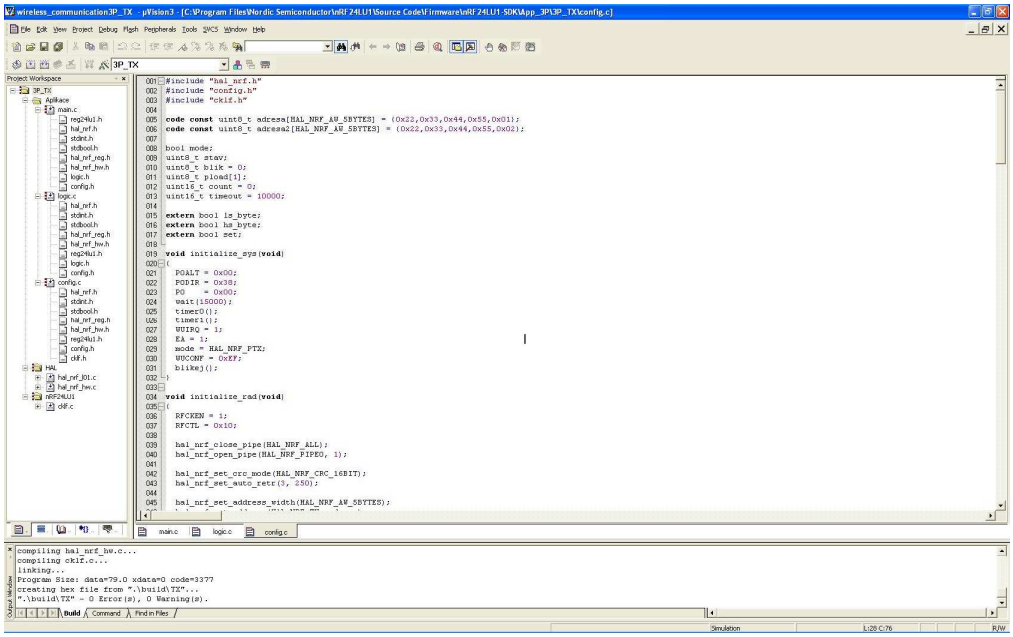


Obrazová příloha č. 4 – Projekt pro primární vysílač.

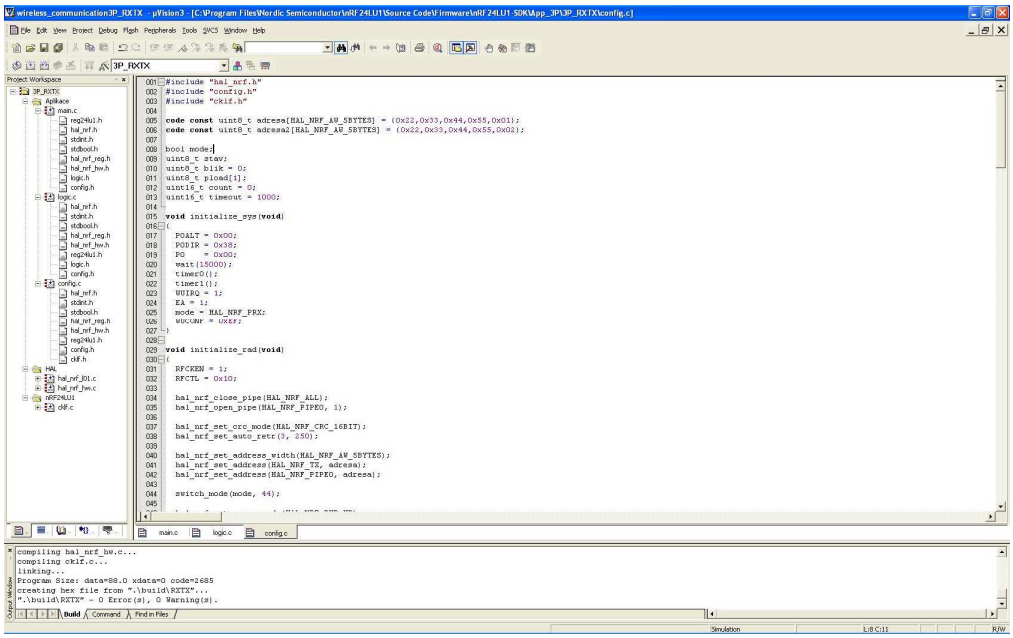


Obrazová příloha č. 5 – Projekt pro primární přijímač.

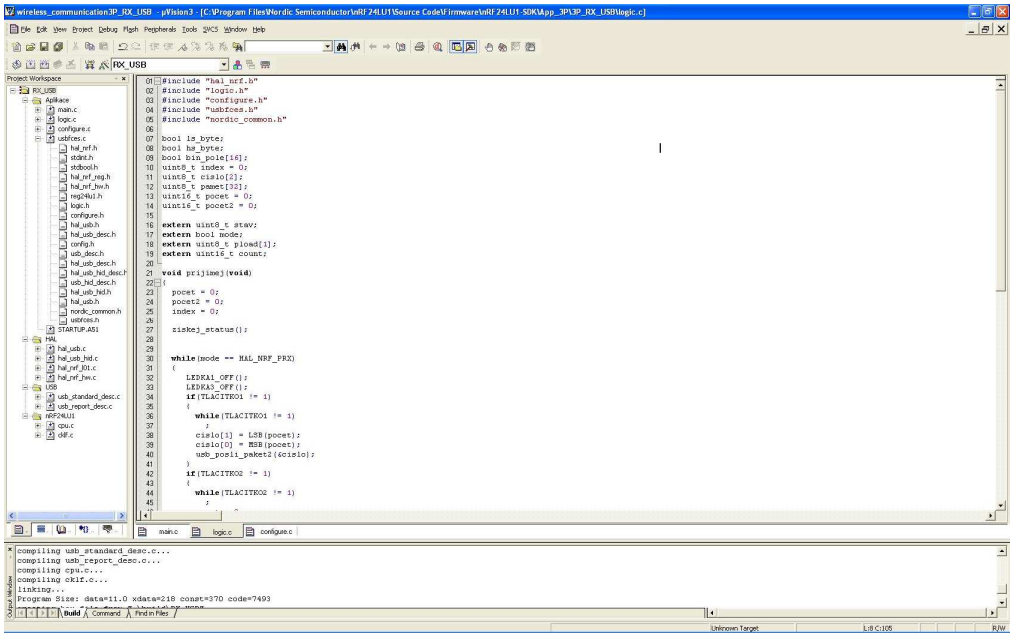
2.2.2 Úloha komunikace tří bodů



Obrazová příloha č. 6 – Projekt pro primární přijímač.



Obrazová příloha č. 7 – Projekt pro střední komunikační bod.

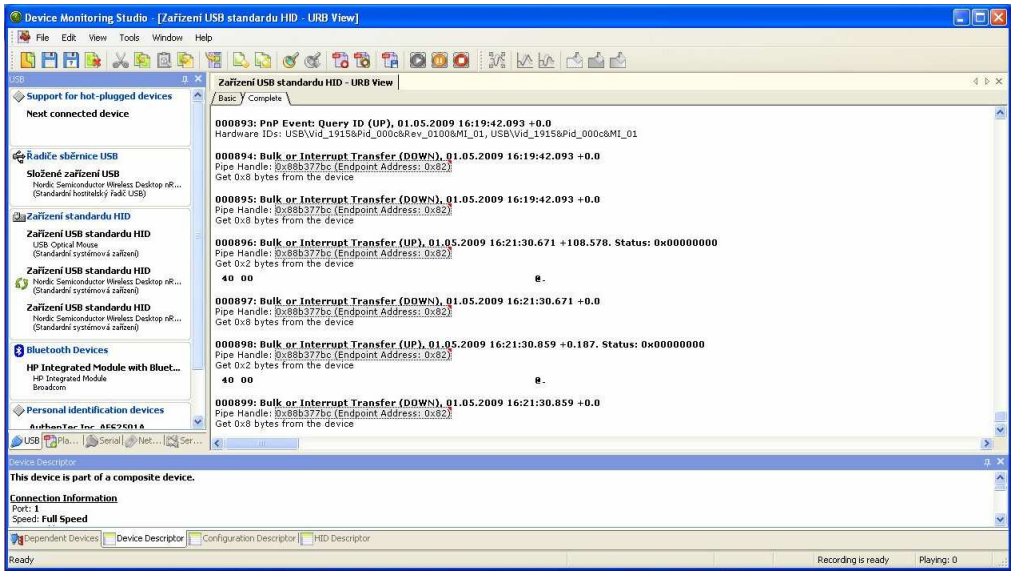


Obrazová příloha č. 8 – Projekt pro primární přijímač.

2.3 Device Monitoring Studio

Device Monitorin Studio je základní nástroj firmy HHD Software pro monitorování USB zařízení a jakýchkoli aplikací které s nimi pracují. Program je kompatibilní s Windows XP. USB Monitor umožňuje příjem, zobrazování, nahrávání a analyzování dat přenášených mezi USB zařízením připojeným k PC a aplikacemi. V bakalářské práci byla využita 14 dení zkušební verze produktu volně poskytovaná ke stažení.

USB Monitor (DMS) [online]. c2009 [cit. 2009-05-01]. Dostupný z WWW: <http://www.hhdsoftware.com/Products/home/usb-monitor.html>.



Obrazová příloha č. 9 – Device Monitorin Studio.